Mise en oeuvre et analyse de solutions de visualisation cartographique sur le web

Rapport de stage

Tanguy THOMAS

Avril - Juin 2018

Remerciements

Je tiens tout d'abord à remercier mes tuteurs, Mme Juliette FABRE et M. Olivier LOBRY, pour la confiance qu'ils m'ont apportés quant à la bonne réalisation de ce projet, ainsi que leur aide et leurs précieux conseils lors du déroulement de ce stage.

Table des matières

Table des figures

Glossaire

Blob Binary Large OBjects est un objet JavaScript qui représente un objet qui contient des données brutes, semblable à un fichier. 20

Boostrap Framework* CSS, HTML et JavaScript créé pour faciliter la création d'une charte graphique d'un site web. 7

Contrôleur Dans le domaine de la cartographie web, ce sont des boutons qui permettent d'intéragir avec la carte pour changer son état ou rajouter des informations (exemple : bouton pour zoomer, échelle..). 8, 19, 20, 26, 27

Framework Ensemble de composants logiciels permettant de créer l'architecture et les grandes lignes d'un logiciel. Se distingue d'une simple librairie par son caractère générique, très peu spécialisé. viii, 7

GeoJson Format d'encodage d'ensemble de données géospatiales simples utilisant la norme JSON (JavaScript Object Notation). Ce n'est pas un format officiel écrit par l'OGC mais par un groupe de développeurs. 28, 37

GetFeature Requête WFS qui récupère des éléments géospatialisés d'une source dans un format donné (Json, XML..). 12, 19

GetMap Requête WMS qui récupère une image d'une couche à un endroit donné. 12

Géocodage Le géocodage est une technique consistant à ajouter des coordonnées géographiques dans une base de données à partir d'une adresse postale.. v, vi, 13, 26, 37, 38, 42

Open Geospatial Consortium L'OGC est une organisation internationale à but non lucratif fondée en 1994 pour répondre aux problèmes d'interopérabilité des Système d'Information Géographique*. 6

Open source La désignation open source, ou « code source ouvert », s'applique aux logiciels (et s'étend maintenant aux œuvres de l'esprit) dont la licence respecte des critères précisément établis par l'Open Source Initiative, c'est-à-dire les possibilités de libre redistribution, d'accès au code source et de création de travaux dérivés. 6, 15, 18

Système d'Information Géographique Un SIG est un système d'information conçu pour recueillir, stocker, traiter, analyser, gérer et présenter des données spatiales et géographiques. viii

Chapitre 1

Introduction

1.1 Présentation générale du contexte

L'Observatoire de Recherche Méditerranéen de l'Environnement (OREME) est une Unité Mixte de Service associant l'Université de Montpellier et des instituts de recherche (CNRS, IRD et Irstea). L'OREME a pour mission la mise en relation d'équipes de recherche dont les activités reposent sur l'observation du milieu naturel.

Il regroupe à ce jour une vingtaine de Systèmes d'Observation (SO) qui génère et récolte des données d'observation. Certains des ces SO sont déclinés en tâches d'observation sur une thématique commune. Par exemple EVOPOP est un SO qui regroupe différentes tâches d'observations sur l'évolution des populations. L'OREME est un Observatoire des Sciences de l'Univers (OSU) qui rallie huit laboratoires de recherches travaillant dans des domaines d'études hétérogènes (géophysique, hydrosciences, écologie, écosystèmes marins, sciences de l'évolution, ...).

Pour étudier des systèmes complexes, les scientifiques doivent être en mesure de croiser des données de domaines divers. C'est à cette problématique que le service "Système d'Information" (SI) de l'OSU OREME s'efforce de répondre. Elle propose ainsi aux équipes scientifiques rattachées à l'Observatoire des services de structuration, normalisation, visualisation, saisie et croisement des données sur des dimensions spatiales, temporelles et sémantiques. Au vu de l'importance cette mission, le stockage de ces données doit être pérenne et accessible au plus grand nombre.

Parmi toutes les données stockées, un grand nombre sont des données géoréférencées qui peuvent être visualisées sur des cartes. Le SI a donc conçu des outils pour visualiser et saisir ces données sur cartes interactives pour le web. Les données cartographiques sont diffusées via un serveur cartographique (GeoServer) et visualisé avec le client cartographique (JavaScript) OpenLayers 2. Par exemple, la carte des stations de l'ancien site minier de Carnoulès montre les emplacements de ces stations de mesures et affiche pour chacune les mesures qui y ont été faites.

1.2 Problématique

Le client cartographique utilisé, OpenLayers 2, n'est plus mis à jour depuis plus de trois ans et manque cruellement de fonctionnalités par rapports aux nouvelles bibliothèques existantes. Le site et les cartes ne sont pas *responsive* et la navigation sur smartphone n'est pas adaptée. De plus l'interface de la carte (boutons, différents contenus textuels) nécessite d'être rafraîchit pour être plus moderne et agréable à utiliser. Le manque de fonctionnalité se fait aussi ressentir sur l'expérience de navigation qui n'est pas complète par rapport à autres sites proposant des cartes interactives.

1.3 Présentation du projet

OpenLayers a subit une refonte complète de son architecture depuis la version 2 et est maintenant à la version 4.6.5. De plus, un concurrent est apparu et a conquis le marché en quelques années : Leaflet. La mission sera donc d'étudier ces bibliothèques en mettant en oeuvre la migration des cartes actuelles et de leurs fonctionnalités avec ces nouvelles bibliothèques, puis les comparer.

Durant ce stage ma mission sera aussi d'être force de proposition afin de suggérer de nouvelles fonctionnalités qui apporteraient un vrai atout aux cartes de l'OREME. Il y aura donc aussi un travail de recherches de toutes les fonctionnalités existantes dans ces deux bibliothèques.

1.4 Annonce du plan

Tout d'abord, je vais vous présenter l'OREME afin d'expliquer leur démarche et l'organisation de cette composante de l'Université. Ensuite je parlerai du résultat de la solution que j'ai produite durant ce stage. Après cela je parlerai du stage d'une façon plus générale, de son déroulement et de mes impressions.

Chapitre 2

Présentation de l'entreprise

L'Observatoire de Recherche Méditerranéen de l'Environnement (OREME) est un Observatoire des Sciences de l'Univers (OSU) consacré à l'observation et l'étude de l'environnement naturel du littoral méditerranéen.

Né en 2007 à l'Université de Montpellier 2 (UM2) à l'initiative de l'INSU, l'OREME est à la fois une Unité Mixte de Service (UMS) et une composante de l'Université de Montpellier. Il est encadré par 4 tutelles :

UM L'Université de Montpellier

CNRS Le Centre National de Recherche Scientifique (CNRS)

IRD L'Institut de Recherche pour le Développement (IRD)

IRSTEA L'Institut National de Recherche en Sciences et Technologies pour l'Environnement et l'Agriculture (IRSTEA)

L'Observatoire concerne 8 Unités Mixte de Recherche (UMR) :

CEFE Le Centre d'Ecologie Fonctionnelle et Evolutive

G-EAU Gestion de l'Eau, Acteurs, Usages

GM Géosciences Montpellier

HSM Hydrosciences Montpellier

ISEM Institut de Sciences de l'Evolution de Montpellier

LUPM Laboratoire Univers et Particules de Montpellier

MARBEC Biodiversité marine et ses usages

TETIS Territoires Environnements, Télédetections et Information Spatiale

L'OREME se focalise sur les risques naturels et l'impact des changements globaux et anthropiques sur l'espace méditerranéen vivant et inerte. Il s'intéresse à tout ce qui se rapporte à l'évolution du changement du milieu naturel méditerranéen. Pour cela, il y a 40 Tâches d'Observation (TO) regroupées thématiquement en 18 Systèmes d'Observation (SO).

2.1 Missions de l'observatoire

L'un de ses principaux challenges réside donc dans sa capacité à récolter, intégrer et partager les données hétérogènes associées à ces disciplines et mettre en évidence des corrélations qui ne pouvaient l'être au préalable. À la clé de cette mise en relation des données : la découverte de signaux systématiques permettant de juger de l'effet du changement global et/ou anthropique et d'en comprendre les mécanismes (aléa, vulnérabilité) dans ses effets environnementaux.

Les missions de l'OREME sont les suivantes :

— soutenir à l'activité ou le développement d'observation systématique en science de l'univers et de l'environnement ;
— soutenir la construction de bases de données environnementales ouvertes, partagées, référencées au niveau international ;
— encourager la mutualisation des moyens analytiques (observation, expérimentation, modélisation) et des savoirs-faire ;
— constituer le relais local des réseaux nationaux d'observations, et un acteur fort des actions tournées vers la Méditerranée en environnement.

2.2 Organisation

L'OREME est dirigé par un directeur : Eric Servat, nommé par le ministre de l'Education nationale sur proposition du conseil de l'OREME. Je travaille à la plateforme : Système d'Information (SI) de l'OREME sous la direction de Juliete Fabre et d'Olivier Lobry (cf Figure 2.1).

FIGURE 2.1 – Organigramme

Chapitre 3

Analyse

Tout d'abord, cette partie présentera l'analyse de l'infrastructure de l'OREME, l'analyse de deux cartes type. Le projet se déroule en plusieurs étapes :

— analyse de l'infrastructure,
— analyse de deux cartes types,
— migration des deux cartes dans les deux librairies,
— comparaison et choix d'une solution,
— implémentations.

3.1 Analyse du sujet et de son contexte

3.1.1 Analyse de l'infrastructure

Infrastructure

Serveur géographique Le Geoserver est un serveur géographique Open source* développé en Java qui permet de diffuser des données géographiques en utilisant les différents protocoles établis par l'Open Geospatial Consortium* :

WMS Web Map Service. Ce protocole propose d'exporter des données géographiques en image.

WFS Web Feature Service. Ce protocole propose d'exporter des données géographiques de manières vectorielles dans différents formats (xml/json..).

WMTS Web Map Tile Service. Ce protocole propose d'exporter des données géographiques en image comme le WMS de manière tuilées.

Base de données Une base de données PostgreSQL stocke toutes les données scientifiques. L'extension PostGIS permet le support de données géoréférencées via un type spécifique "*geometry*" supportant les éléments (point, polygone et ligne) ainsi que des fonctions spatiales. Les données de la base peuvent être exportées vers Geoserver sous forme de couches (*layers*) diffusables via ces différents protocoles.

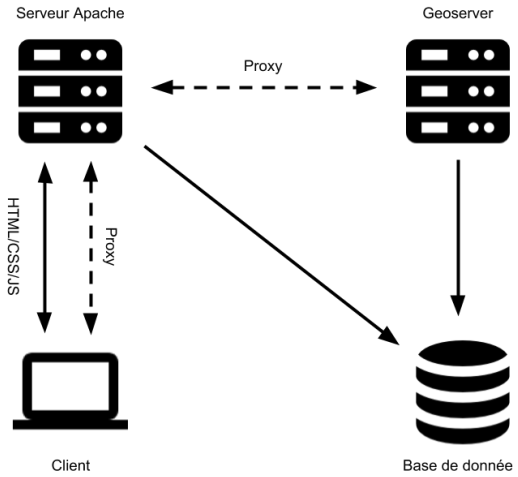

FIGURE 3.1 – Infrastructure

Web Le site `data.oreme.org` est le portail de données de l'OREME. Il est développé en PHP à l'aide du Framework* CodeIgniter, le style est géré avec Boostrap*. Les données sont organisées en Service d'observation (SO) et tâches d'observation (TO). Des cartes interactives sont mises en place pour chacune de ces thématiques (librairie OpenLayers 2).

Description de deux cartes

L'OREME m'a demandé d'étudier et de migrer deux cartes qui représentent au mieux l'éventail de fonctionnalités utilisées par l'OREME que je listerai plus tard (cf section 3.2, page 11)

Carte des stations de de mesure du site minier de Carnoulès Cette carte représente les stations de mesure d'une tâche visant à étudier l'évolution des paramètres pshysico-chimiques de rivières en aval de l'ancien site minier de Carnoules (Gard). C'est la première que j'ai reproduite puisqu'elle n'utilise presque que le tronc commun des fonctionnalités entre toutes les cartes.
Cette carte affiche deux couches de stations de mesures, les stations publiques en rouge et privées en bleu. Les fonds de carte proposés sont : OpenStreetMap (par défaut), Google Map, Google Sat. Les stations sont interrogées avec le protocole WFS sur le serveur cartographique. L'utilisateur peut cliquer sur chaque point de la carte pour sélectionner la station (affichage en jaune), et afficher ses

informations sur le côté droit de la carte. Ces informations sont récupérées dynamiquement en Ajax à l'aide de l'identifiant de la station concernée (cf Figure 1).

Quatre Contrôleurs* sont proposés :

— Le 'layer switcher' qui permet de changer le fond de carte sélectionné (*Base layer*) et d'enlever ou rajouter les couches qui sont sur la carte (*Overlays*).
— Deux boutons de zoom,
— Une échelle
— Une *attribution* de carte, ici 'Data CC-By-SA by OpenStreetMap', qui permet de décrire le fournisseur du fond de carte.

Une légende est affichée sous la carte. Elle est obtenue avec la requête 'WMS GetLegendGraphic' dont le fonctionnement est expliqué plus loin (Section 3.2.2, page 13).

Carte des stations de mesure des truites communes Cette carte représente les stations d'une tâche visant à étudier l'évolution génétique d'échantillonage des populations de truites. Elle est bien plus complexe que la première. En effet elle utilise plusieurs technologies différentes et propos d'autres fonctionnalités que j'énumérerai après.

La carte des stations d'échantillonage des truites comporte les mêmes Contrôleurs*, les même fond de cartes, et une couche de stations importée via le protocole WFS. Cette carte affiche également le réseau hydrographique français, cette couche est interrogée avec le protocole WMS (BD Carthage de l'IG (cf Figure 2)). Ces couches correspondent à des niveaux de détails d'une couche d'entités hydrographiques. En fonction du niveau de zoom, la carte n'affiche pas le même niveau de détail des entités hydrographiques. Ce système à été implémenté afin de ne pas surcharger la carte, autant au niveau visuel qu'au niveau des performances. La figure 3.2 montre quatre captures d'écran de la carte avec plusieurs niveaux de zoom différent.

Un onglet intitulé "Options d'interrogation" permet de choisir quel élément de la carte on interroge, soit les stations, soit les entités hydrographiques (lacs et rivières). Ce choix est indispensable puisque OpenLayers 2 n'est pas capable de gérer les deux interactions à la fois du protocole WMS et du WFS.

Stations Affiche sur le côté des informations sur le point sélectionné et change l'affichage du point sur la carte. (Figure 1).

Cours d'eau Affiche un pop-up avec des informations sur le cours d'eau sélectionné et affiche une couche avec le cours d'eau sélectionné en rouge (Figure 3.3).

La carte propose aussi la fonctionnalité de filtrer les stations affichées sur la carte (cf onglet Critères de sélection des stations sur la Figure 2). Ce sont des filtres dynamiques qui sont exécutés via une requête Ajax qui récupère les stations à afficher.

FIGURE 3.2 – Carte des truites communes à des niveaux de zoom différents - OpenLayers 2

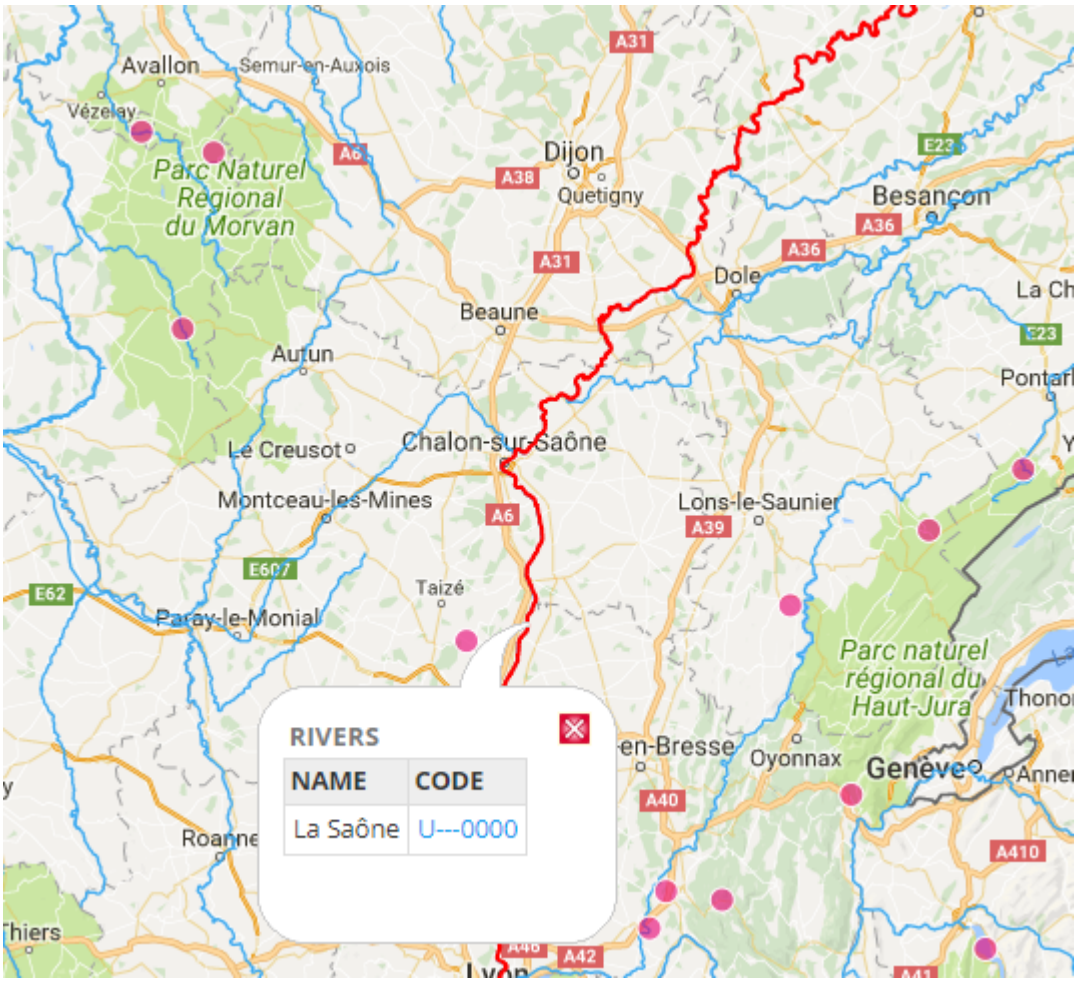

FIGURE 3.3 – Carte des truites communes - Sélection d'un cours d'eau

3.1.2 Analyse de l'existant

Les bibliothèques de cartographie web

Deux bibliothèques de cartographie web en JavaScript sont dominantes sur le marché, ce sont les seules à proposer autant de fonctionnalités :

OpenLayers Maintenant dans sa version 4.6, Openlayers est plutôt destiné à une utilisation très poussée de la cartographie. C'est une bibliothèque qui intègre beaucoup de fonctionnalitées et supporte les principaux protocoles établis par l'OGC

Leaflet Leaflet propose moins de fonctionnalités qu'OpenLayers mais un système de plugins (greffons) à rajouter à la bibliothèque principale. Bien que créée plus tard, elle est maintenant plus utilisée sur le web qu'Openlayers. Elle est beaucoup appréciée pour sa simplicité d'utilisation et de mise en place.

3.2 Analyse des besoins fonctionnels

Cette partie présente les différentes fonctionnalités à analyser et à mettre en place. Une partie de ma mission est de trouver des fonctionnalités intéressantes et utiles à rajouter sur les cartes, le cahier des charges a évolué à travers le temps. Ces fonctionnalités sont regroupées en trois parties, les fonctionnalités indispensables, dispensables, et celles qui ont étés découvertes au fur à mesure de mon analyse.

3.2.1 Fonctionnalités indispensables

Proxy

Le serveur cartographique est accessible via un autre domaine que celui du site. Cela pose des soucis de *cross-origin* (inter-domaine), pour palier ce problème, l'OREME utilise un proxy qui se trouve à l'URL `https://data.oreme.org/cgi-bin/proxy.cgi` auquel on fournit via la méthode GET le paramètre `url` qui contient l'url d'un domaine différent.

Protocoles standardisés de l'OGC

L'OREME utilise les protocoles WMS et WFS afin de diffuser ses couches et d'interroger d'autres serveurs géographiques publiques. Ces protocoles sont des standards très utilisés et garantissent un maximum d'interopérabilité. La bibliothèque utilisée devra impérativement supporter ces protocoles.

Export en jpg/pdf

Beaucoup de chercheurs veulent avoir la possibilité d'exporter des cartes en images pour les partager facilement. Afin d'améliorer l'expérience utilisateur, l'image exportée

devra contenir les potentiels labels et autres objets de la carte, ainsi qu'une légende.

Style

La bibliothèque devra donner la possibilité de styliser des éléments de la carte (généralement pour le protocole WFS). Par exemple de changer la couleur de l'élément et de sa bordure, la taille de l'élément etc..

Permalien

Un permalien (*permalink*) permet d'enregistrer l'état de la carte sous la forme d'une URL afin de pouvoir la partager. L'URL devra contenir par exemple :

- le positionnement sur la carte :
  - le niveau de zoom de la carte ;
  - les coordonnées du centre (lat/lng) ;
  - la rotation de la carte.
- l'élément actuellement sélectionné sur la carte,
- le fond de carte actuel,
- d'autres informations spécifiques à déterminer au cas par cas.

Interactions avec la carte

Pour chaque protocole utilisé, il devra être possible d'interagir avec les éléments de la carte. Autant en WMS qu'en WFS, la possibilité d'interroger la carte au clic est indispensable. Les couches en WFS devront supporter la gestion d'événements et les couches WMS devront supporter l'envoie de requête 'GetFeatureInfo' lors d'un clic sur la carte.

Cohabitation des interactions WMS/WFS

OpenLayers 2 ne gère soit une interaction pour une couche WMS, soit une interaction pour des couches en WFS, mais ne peux pas gérer les 2 en même temps. Il faut donc que l'utilisateur choisisse un des deux interactions il veut via un onglet 'Option d'interrogation'. Il serait donc intéressant de trouver un moyen de ne plus avoir cet onglet qui est une contrainte pour l'utilisateur.

Filtre

Les couches doivent pouvoir être filtrées dynamiquement comme les protocoles WMS et WFS le permettent lors d'une requête 'GetMap'* et 'GetFeature'*. Cette fonctionnalité est essentielle dans plusieurs cartes déjà présentes et devra être intégrée dans la bibliothèque choisie.

3.2.2 Fonctionnalités additionnelles

Légende

Il serait intéressant de bénéficier d'une légende qui aide à la compréhension de la carte. Elle devra être dynamique en fonction des différentes couches affichées sur la carte. Actuellement le protocole WMS permet via une requête 'GetLegendGraphic' de générer une légende en image. Or l'image générée n'est pas très élégante ni harmonieuse sur la carte, il s'agirait de trouver une alternative plus esthétique.

Cartes thermique (*Heatmap*)

Une carte thermique est une représentation graphique de données statistiques géospatialisées. Elle permet, par exemple, d'afficher la densité d'une espèce de scarabée à un moment donné sur la métropole. Les cartes de chaleur propose un moyen de visualiser simplement beaucoup d'informations et d'analyser certains comportements.

Couches temporelles

Certaines cartes peuvent intégrer une troisième dimension, celle du temps. Elles sont utilisées pour montrer l'évolution d'un ou plusieurs éléments géospatialisés, par exemple, l'évolution d'un trait de côte à travers le temps.

Altitude

Tout comme les couches temporelles, on peut intégrer une troisième dimension, celle de l'altitude.

Géolocalisation

Certaines cartes de l'OREME n'ont pas pour but de diffuser mais de saisir de la donnée liée à un emplacement. Pour faciliter cette saisie, l'intégration d'un bouton de géolocalisation sur la carte permet de ne pas avoir à chercher l'emplacement de la mesure si l'on s'y trouve.

Géocodage*

Le Géocodage* (*geocoding*) est la technique qui permet de récupérer des informations géographiques à partir d'une adresse ou d'un nom de lieu. Pour certaines cartes avec de nombreux points sur une grande étendue géographique, ou pour de la saisie de données, cette fonctionnalité a un réel intéret.

Regroupement de points

Le regroupement de points géographiques (*cluster*) est le principe d'agglomérer plusieurs points assez proches en un seul point afin d'alléger et de rendre plus lisible la

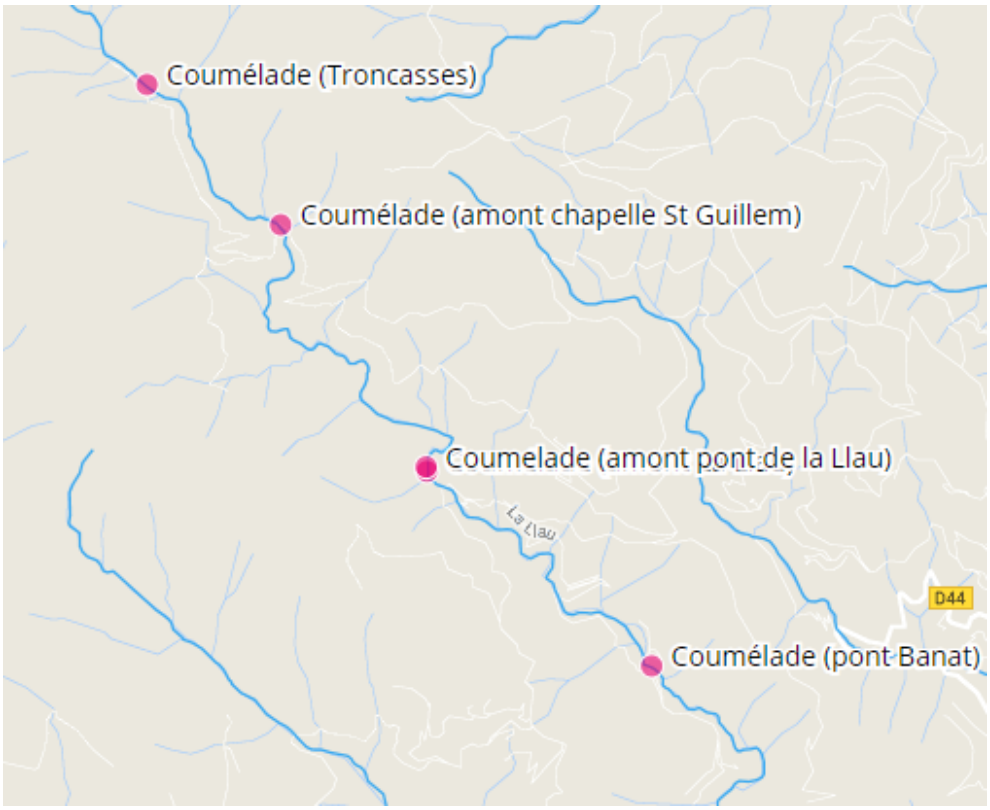

FIGURE 3.4 – Cartes des truites communes - Labellisation des stations

carte. Cette fonctionnalité est utile lorsqu'il y a beaucoup de points, par exemple pour la carte des truites communes qui regroupe plus de mille points.

Annotations graphiques

Beaucoup de chercheurs utilisent les cartes de l'OREME dans leurs rapports, il serait donc intéressant de pouvoir dessiner sur la carte ; cela pourrait être un ajout intéressant pour l'expérience utilisateur.

Pop-Up

Les pop-ups sont utilisés actuellement pour afficher des informations lorsque l'utilisateur clique sur un cours d'eau (Figure 3.3). C'est donc un élément indispensable que devra fournir la bibliothèque.

Labellisation des points

L'affichage d'informations textuelles sur les cartes était une fonctionnalité demandée par plusieurs chercheurs, et elle est ainsi en option sur la carte des truites par exemple (Figure 3.4).

3.2.3 Diagramme UML

Le diagramme de cas d'utilisation (cf Figure 3.5) montre toutes les actions que peut faire un utilisateur pour la carte des stations de mesures des truites communes.

3.3 Analyse des besoins non-fonctionnels

3.3.1 Spécifications techniques

Langage

Afin de réaliser des cartes Web interactives, il faut utiliser un langage de programmation web côté client : le JavaScript. Par ailleur, il est possible que j'utilise aussi le langage PHP côté serveur.

Open source*

Les valeurs que représente l'open source sont importantes au sein de l'équipe. De plus l'open source a des avantages non négligeable, comme la gratuité, la possibilité de modifier et de voir le code source.

Adaptabilité (*Responsive design*)

Depuis maintenant deux ans, le nombre de consultations Internet a été plus élevé sur les équipements mobiles que sur les PC. Du fait de cette part importante d'utilisateur sur mobiles et de son augmentation constante, il est impératif d'avoir un site *responsive*.

Plein Ecran

Les cartes du site de l'OREME doivent aussi afficher d'autres informations, les cartes sont donc petites, autant sur mobile que PC. La possibilité de mettre la carte en plein écran améliore l'expérience utilisateur.

3.3.2 Structuration du code

Dans le code des cartes déjà existantes, une grande partie du code est redondante et pourrait être largement réduite à l'aide de fonctions qui automatiseraient certaines créations d'éléments :

La carte La création de la carte est essentiellement courte, mais les contrôleurs qui lui sont ajoutés sont généralement les mêmes sur chacune des cartes, de même pour les fonds de carte.

Les couches WFS La création d'un *layer* WFS est assez longue et peut être réduite à une fonction qui prend l'entrepôt, le nom de la couche sur Geoserver et

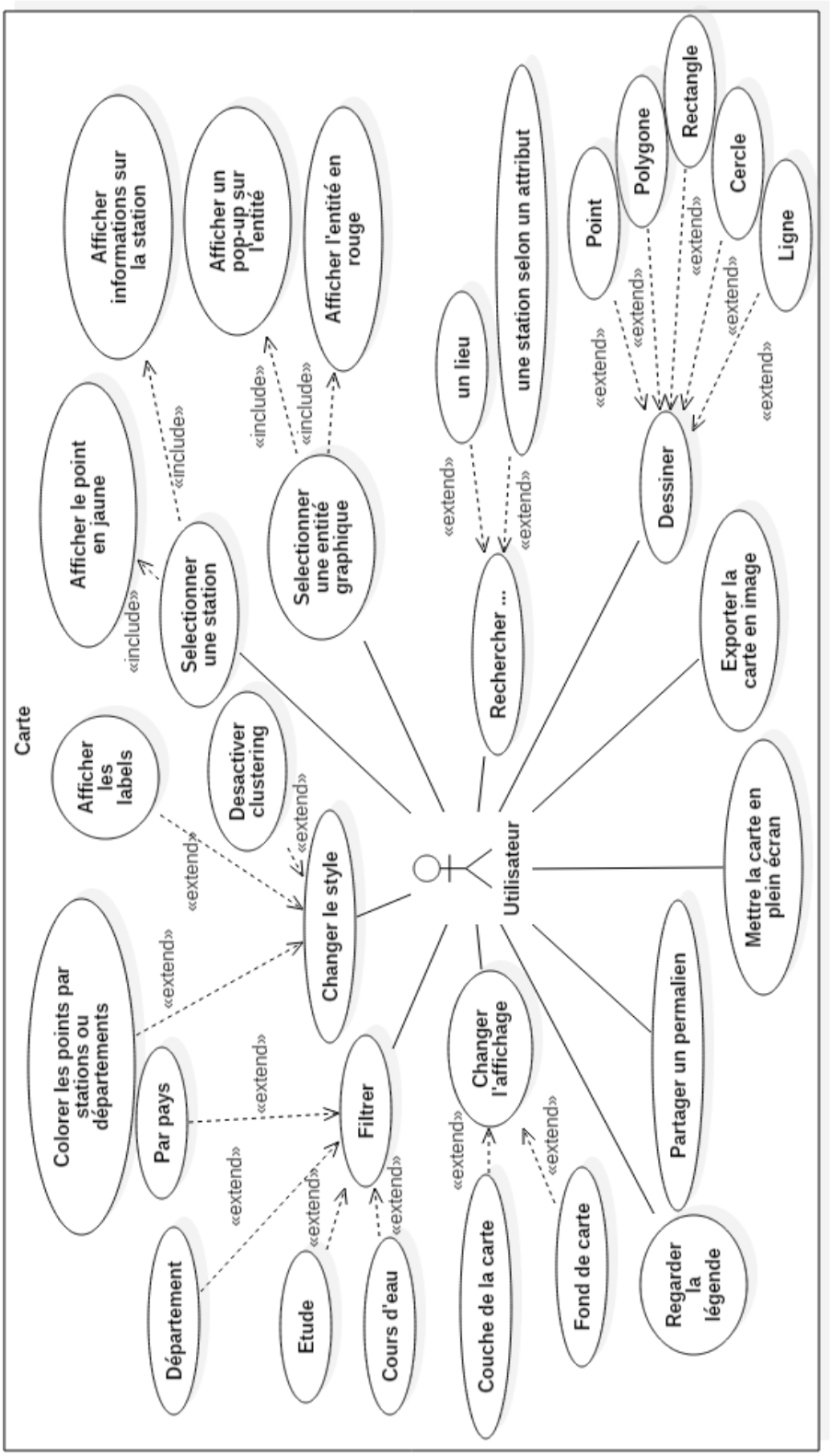

FIGURE 3.5 – Diagramme de cas d'utilisation - Truites communes

une fonction à appeler lors d'un clic. En options l'utilisateur pourrait potentiellement rajouter des filtres.

3.3.3 Documentation

L'OREME va utiliser mes travaux dans le futur lors de création de nouvelles cartes et pour finir la migration des cartes actuelles ; il est donc important de bien commenter et documenter le code pour les autres personnes qui vont l'utiliser. Ce rapport est aussi destiné à l'OREME pour expliquer ma démarche et comment chaque fontionnalité est implémentée.

Chapitre 4

Rapport technique

Cette section rend compte de l'étude des deux bibliothèques dans le cadre de la migration des cartes actuellement en OpenLayers 2. Chacune des fonctionnalités sera analysée, leur comportement, complexité d'installation, avantages et limites. Finalement une synthèse de cette analyse sera rendue.

4.1 Analyse d'Openlayers 4

4.1.1 Général

OpenLayers 4 est un des pionniers dans la cartographie Web Open source*. La version 2 de la bibliothèque est sortie en 2006, avec une refonte complète de l'architecture via la version 3, OpenLayers semble maintenant être une bibliothèque mature, réfléchie et bien construite. Cette bibliothèque est plutôt utilisée par des professionnels grâce à un support natif des protocoles utilisés par des serveurs cartographiques.

Le site d'OpenLayers propose une documentation complète et une section qui regroupe plus de 150 exemples afin de guider les développeurs qui découvrent la bibliothèque. Avec la grande quantité de fonctionnalités que propose OpenLayers, ce support est très utile pour la conceptions de carte. La bibliothèque compte plus de 150 classes différentes, ce qui rend la conception très lourde. Il faut utiliser beaucoup de classes pour faire des actions simples. Toutefois cela permet d'avoir un code explicite qui se comprend facilement.

4.1.2 Support du WMS

La bibliothèque propose de créer une couche via le protocole WMS en lui fournissant l'URL et des paramètres de la requête WMS :

- nom(s) de la ou les couches que l'on veut récupérer (string avec chaque couche séparée par une virgule),
- la version du protocole utilisé,
- si l'image doit être transparente ou non,

— le format de l'image (seul le format png est transparent),
— un potentiel filtre si l'on veut afficher qu'une partie des informations de la couche,
— ..

L'image est récupérée via une URL construite à partir de l'URL du service WMS du Geoserver et des paramètres de la requête donnés via la méthode GET. Afin de pouvoir utiliser le proxy, une méthode de l'objet instancié à été redéfinie pour changer l'URL vers celui du proxy.

La requête 'GetFeatureInfo' elle doit être gérée à la main. Un gestionnaire d'évènement de type 'singleclick' est ajouté à la carte, la latitude et longitude de l'endroit cliqué sont récupérées et une requête ajax vers Geoserver est envoyée. La réponse est affichée dans un pop-up de sur la carte s'il y a des informations à afficher.

4.1.3 Support du WFS

Une classe permet de créer une requête 'GetFeature'* avec des paramètres données. Ainsi on a aussi la possibilité d'ajouter des filtres à la requête afin de sélectionner seulement certaines données géographiques. Une requête ajax (POST) peut être envoyée au Geoserver avec la requête traduite en XML 'GetFeature'* comme données. De cette manière le proxy peut être mis en place très simplement puisque OpenLayers ne créé pas lui même l'URL finale mais c'est au développeur de faire la requête ajax.

4.1.4 Cohabitation des interactions WMS/WFS

Le fonctionnement des options d'interrogation a été entièrement repensé depuis OpenLayers 2 et la nouvelle manière de faire permet de base la cohabitation native des interactions avec la carte.

4.1.5 Responsivité

OpenLayers 4 est maintenant responsive et s'adapte très bien sur smartphone ou tablette. La navigation sur les cartes est très fluide et réactive. OpenLayers réagit bien à une utilisation à un et à deux doigts sur la carte, y compris la rotation, qui est gérée nativement.

Cependant, l'icône du Contrôleur* qui met la carte en pleine écran ne semble pas être adapté sur mobile puisqu'il est modifié et ne ressemble plus à l'icône de base, il devra être modifié.

4.1.6 Plein écran

Un Contrôleur* permet d'avoir un bouton pour afficher la carte en plein écran. De plus il est possible de choisir exactement l'élément HTML du document qui sera mis en plein écran. On peut donc choisir un élément parent à la carte afin d'afficher des

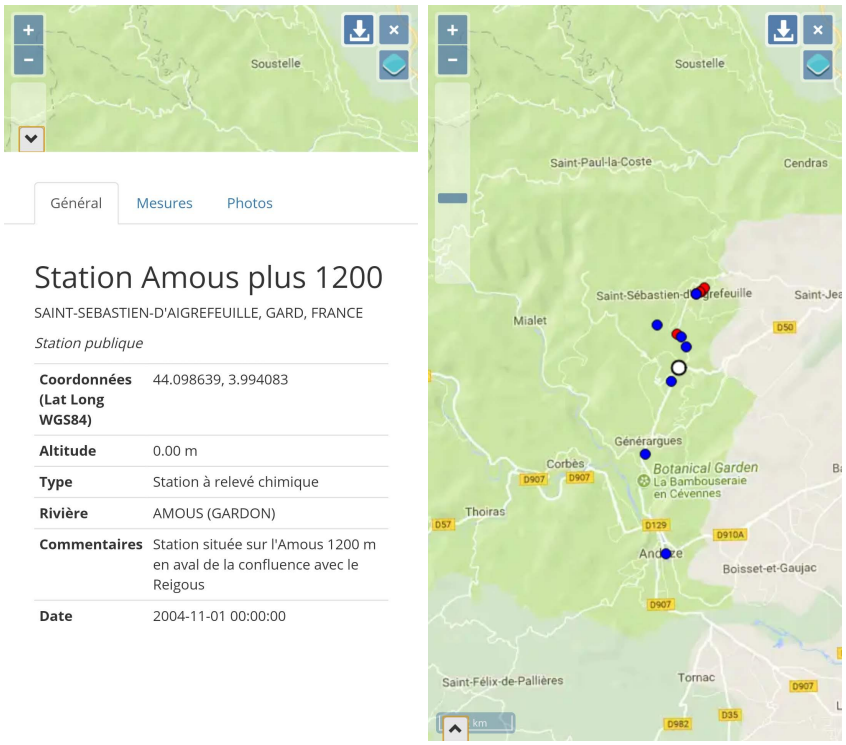

FIGURE 4.1 – OpenLayers 4 - Plein écran - Panneau d'information ouvert / fermé

informations avec la carte. C'est cette fonctionnalité que j'ai utilisé afin d'afficher les informations sur les stations sélectionnées (cf Figure 4.1).

4.1.7 Exportation en jpg/pdf

OpenLayers 4 ne permet pas d'exporter une carte de manière native. Cependant la carte est représentée à l'aide d'un `canvas` HTML et JavaScript intègre un méthode pour générer un Blob* qui représente `canvas` en une image que l'on peut exporter et télécharger. Cette méthode me permet donc, via un Contrôleur* customisé, de faire un bouton afin de générer une image de la carte (cf Figure 4.2).

Figure 4.2 : Ce code crée la classe `DownloadImgControl` qui hérite de la classe `ol.control.Control`. Cette classe permet donc de créer un Contrôleur*, qui exécutera la fonction `ddl` afin de lancer le téléchargement de la carte. La conception de cette fonction est compatible avec la plupart des grand navigateurs existants.

4.1.8 Permalien

Le site d'OpenLayers propose un exemple de code qui permet de générer un *permalink* de la carte. Ce code est à adapter avec chaque spécification de chaque carte.

```
app.DownloadImgControl = function (opt_options) {
let options = opt_options || {};
let ddl = function () {
    map.once('postcompose', function (event) {
        var canvas = event.context.canvas;
        if (navigator.msSaveBlob) {
            navigator.msSaveBlob(canvas.msToBlob(), 'map.png');
        } else {
            canvas.toBlob(function (blob) {
                saveAs(blob, 'map.png');
            });
        }
    });
    map.renderSync();
};
let button = document.createElement("button");
button.addEventListener("click", ddl);
button.title = "Telecharger la carte (png)";
button.innerHTML = "<i class=\"glyphicon glyphicon-download-alt
   \"></i>";
let element = document.createElement("div");
element.className = 'ol-ddl ol-unselectable ol-control';
element.appendChild(button);
ol.control.Control.call(this, {
    element: element,
    target: options.target
});
};
ol.inherits(app.DownloadImgControl, ol.control.Control);
```

FIGURE 4.2 – OpenLayers 4 - Code - Exportation de la carte

```
let updatePermalink = function () {
if (!shouldUpdate) {
    shouldUpdate = true;
    return;
}
let center = view.getCenter();
let url = '#map=' +
    view.getZoom() + '/' +
    Math.round(center[0] * 100) / 100 + '/' +
    Math.round(center[1] * 100) / 100 + '/' +
    view.getRotation() + '/' +
    selected + '/' +
    idlayerBase;
let state = {
    zoom: view.getZoom(),
    center: view.getCenter(),
    rotation: view.getRotation(),
    id: selected
};
window.history.pushState(state, 'map', url);
};
```

FIGURE 4.3 – OpenLayers 4 - Code - Permalink

A chaque mouvement de la carte les informations suivantes sont enregistrées dans l'URL :

- les paramètres d'affichage de la carte (niveau de zoom, coordonnées du centre et la rotation) ;
- le fond de carte ;
- l'élément actuellement sélectionné sur la carte.

Ce mécanisme fonctionne via trois fonctions :

- une qui enregistre les données dans l'URL : `updatePermalink()` ;
- une qui récupère les données de l'URL au chargement de la page ;
- une qui change l'état de la carte lorsque l'utilisateur revient en arrière via son navigateur.

updatePermalink Figure 4.3 : cette fonction créée un *string* : `url`, qui regroupent les différentes informations à transmettre dans l'URL. Un objet JavaScript, `state`, est aussi créé avec ces mêmes informations. Ces deux éléments sont utiles à la création d'une nouvelle entrée dans l'historique du navigateur qui modifie l'URL en rajoutant le *string* en *hash* de l'url. Cette fonction est appelée à chaque modification de l'état de la carte.

Récupération des données de l'url au chargement de la page A la fin du chargement de la page, le contenu du hash est récupéré et analysé afin d'obtenir les

informations pour d'initialiser la carte (cf Figure 4).

Changement de l'état de la carte Dans la fonction `updatePermalink`, la méthode `pushState`, qui permet de rajouter un état à l'historique d'une page web, est utilisée afin d'enregistrer l'état actuel de la carte. Lorsque l'utilisateur fait un retour en arrière sur la page web, un évènement de type `'popstate'` est lancé. Une fonction est appelée à chacun de ces évènements grâce à un gestionnaire d'évènements : elle va changer l'état de la carte à l'aide des informations créées dans la fonction `pushState` (objet `state`) (cf Figure 5).

4.1.9 Légende

OpenLayers 4 ne propose aucun moyen de créer une légende, ni à partir d'une couche WMS ni à partir d'un style appliqué par la bibliothèque elle-même.

4.1.10 Couches Temporelles

OpenLayers ne supporte pas les couches temporelles. Cependant il est possible de récupérer simplement la couche à un temps précis en rajoutant le paramètre TIME à la requête du Geoserver.

4.1.11 Cartes thermique (Heatmap)

Une classe OpenLayers permet de générer une carte thermique. On peut personnaliser le poids d'un point de la carte (valeur entre 0 et 1 qui représente la "chaleur" du point). On peut aussi personnaliser le gradient de couleur de la carte via un tableau de couleurs CSS ('white', 'rgb(0,0,0,1)', '#00f'..) (Figure 4.4).

4.1.12 Altitude

OpenLayers 4 ne propose toujours pas de moyen d'afficher des couches avec de l'altitude.

4.1.13 Regroupement de points

Il existe un moyen de générer des *clusters* dans OpenLayers 4, cependant celui impose de créer soit même le style du point de cluster, ainsi que toutes interactions avec ce dernier, si l'on veut par exemple que lorsque l'on clique sur un cluster la carte zoom sur le contenu du cluster, il faudra le faire soi-même. La figure 4.5 montre un exemple très simple de design de cluster. Il serait aussi intéressant pour les utilisateurs de leur donner la possibilité de desactiver le *clustering* à leur guise.

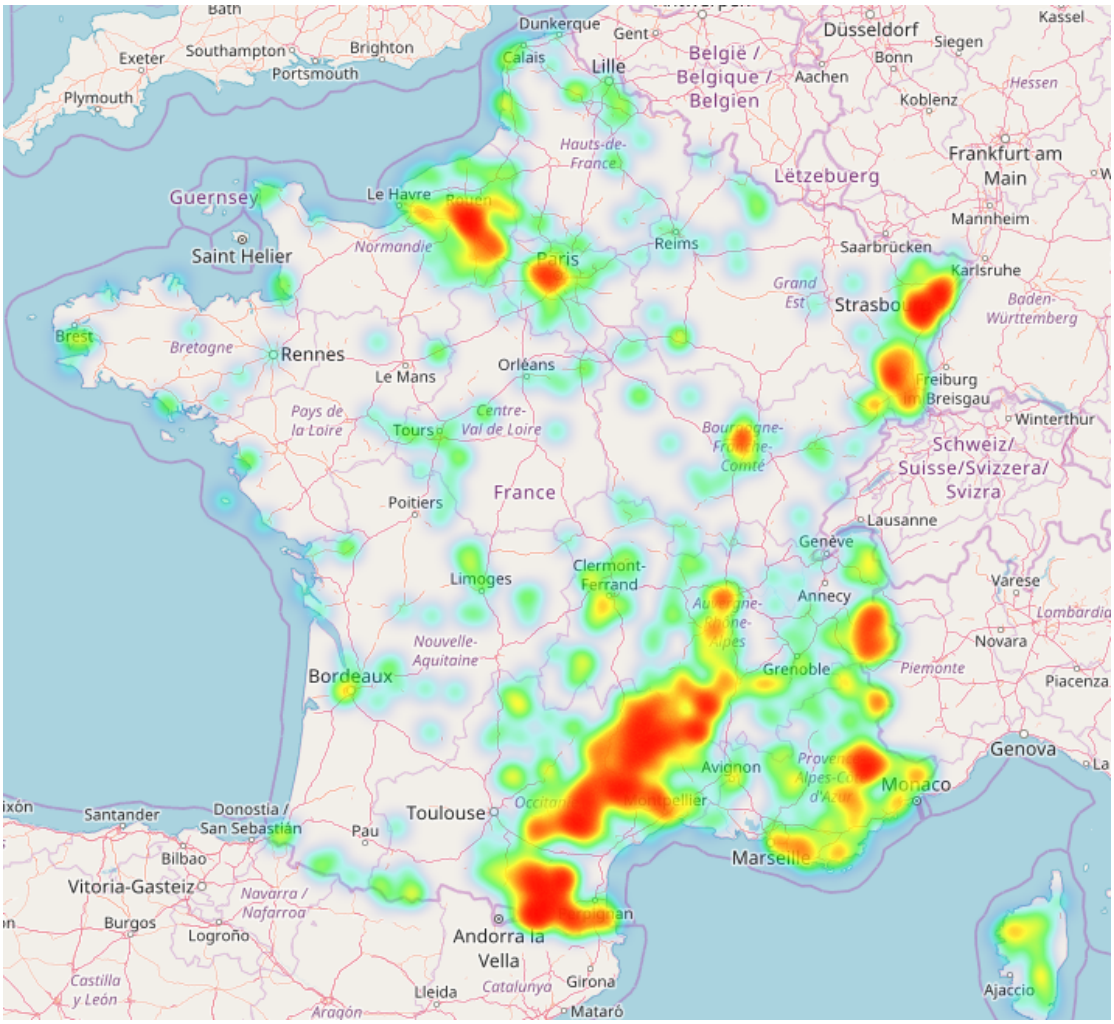

FIGURE 4.4 – OpenLayers 4 - Heatmap

FIGURE 4.5 – OpenLayers 4 - Exemple de regroupement de points

```
var geocoder = new Geocoder('nominatim', {
provider: 'osm', // Fournisseur = OpenStreeetMap
lang: 'fr-FR', // Langue Francaise
placeholder: 'Recherche pour ...',
targetType: 'text-input',
limit: 5, // Limite de proposition de l'auto-completion
keepOpen: false // Rendre le controle retractable
});
map.addControl(geocoder);
```

FIGURE 4.6 – OpenLayers 4 - Code - Geocoder

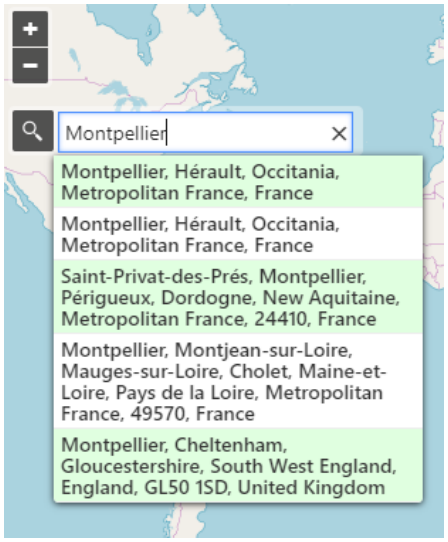

FIGURE 4.7 – OpenLayers 4 - Contrôleur Geocoder

4.1.14 Géocodage*

`https://github.com/jonataswalker/ol-geocoder`

Un module, OpenLayers Control Geocoder, propose d'implémenter un Contrôleur* qui permet de faire du Géocodage*. Le module propose plusieurs options utiles dont le choix du fournisseur (*provider*) de données. Dans une optique d'utiliser le plus de service libre, le service de *geocoding* est OpenStreetMap (cf Figure 4.6). Un système d'auto-complétion est aussi implémenter afin de faciliter la recherche.

Il rajoute un Contrôleur* qui est en fait une champ de recherche rétractable dans laquelle on écrit le nom du lieu recherché (Figure 4.7). Cependant, le zoom se fait sur un point et non un polygone. Par exemple, si l'utilisateur cherche la ville de Montpellier, la carte va zoomer au centre de la ville au lieu de zoomer sur toute la surface de la ville.

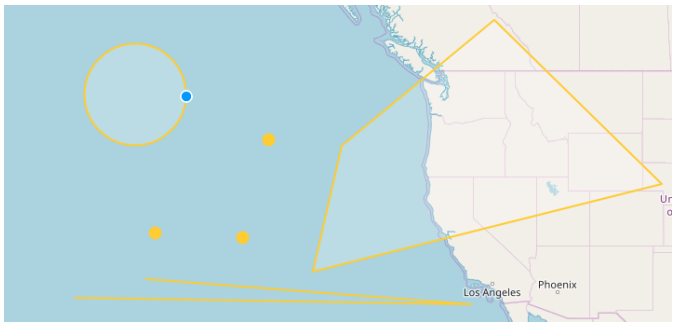

FIGURE 4.8 – OpenLayers 4 - Annotation graphique

4.1.15 Annotation graphique

OpenLayers intègre des classes qui permettent d'ajouter des annotations graphiques sur la carte. Cependant le support de ces interactions est très basique. Il n'y a aucun Contrôleur* pour choisir le type d'élément que l'on veut ajouter et il est impossible de modifier ou supprimer une annotation graphique de la carte après l'avoir créé (Figure 4.8).

4.1.16 Pop-Up

Sur OpenLayers 4, on peut afficher un *pop-up* un *overlay* sur la carte qui affiche un élément HTML à une coordonnée précise. On peut styliser en CSS l'élément HTML créé.

4.1.17 Labellisation des points

OpenLayers propose dans la classe `ol.style.Style` (classe qui gère le style d'un élément de la carte) d'intégrer un objet `ol.style.Text` qui de manière très complète permet de :

— gérer la position et rotation du texte par rapport à l'élément,
— changer la couleur, taille, bordure,
— police d'écriture...

La labellisation est possible et personnalisable très simplement sur n'importe quel élément de la carte (lignes, points, polygones).

4.2 Analyse de Leaflet

Leaflet est une bibliothèque de cartographie web créée en 2011. Elle est donc bien plus jeune que OpenLayer et a aussi une idéologie complètement différente. Leaflet est très légère (38kb) et intègre peu de fonctions de base mais privilégie l'ajout de plugin créé par la grande communauté autour de Leaflet pour rajouter de nouvelles

fonctionnalités. Leaflet possède une architecture plutôt simple qui rend la rédaction du code courte et claire.

4.2.1 Protocole WMS

Module de support du protocole WMS : `https://github.com/heigeo/leaflet.wms`

Leaflet propose nativement un support pour les couches WMS, mais il est assez restreint. Il ne gère que les couches tuilées à l'aide de la classe native `L.tileLayer.wms`. De plus, il n'y a pas de méthode qui gère le 'GetFeatureInfo'. Le *plugin* `leaflet.wms` apporte un support plus complet au protocole WMS puisque, celui natif est trop limité.

Le module, bien que très complet, ne supporte pas l'utilisation d'un proxy. Cependant, il est possible de personnaliser des méthodes du module qui gère la récupération et l'affichage des informations. J'ai redéfini ces méthodes de manière à permettre l'utilisation du proxy, l'affichage des informations dans un *pop-up* (Figure 4.23) et d'un layer supplémentaire représentant l'entité hydrographique sélectionnée avec un style différent (rouge et large).

4.2.2 Procole WFS

Panneau latéral (*sidebar*) : `https://github.com/Turbo87/leaflet-sidebar`

A l'aide d'une requête ajax envoyé au Geoserver, il est possible de récupérer des données géographiques en GeoJson*. Leaflet est capable de lire et traduire le GeoJson en un objet de Leaflet représentant une couche.

Leaflet propose un support des événements sur chaque élément d'une carte. On peut ajouter un gestionnaire d'évènement sur n'importe quel type d'évènement. Une fonction est alors déclenchée lorsque l'utilisateur clique sur un élément
`L.DomEvent.on(stations,"click",info)`.
La fonction `L.DomEvent.on` permet d'attacher la fonction `info` lorsque l'on clique sur un élément de la couche `stations`.

Sur la figure 4.10, l'utilisateur a cliqué sur un marqueur, qui s'est coloré en jaune afin d'indiquer qu'il est sélectionné. Cela déclenche aussi l'apparition d'un panneau latéral avec des informations sur la station sélectionnée. Ce panneau latéral est un contrôleur personnalisé qui prend toute la hauteur de la carte. Les méthodes `show` et `hide` font respectivement apparaître et disparaître le panneau.

Une interaction permet de sélectionner un élément de la carte qui appelle une fonction *callback* à chaque sélection/désélection de la carte. Leaflet gère uniquement les événements natif de JavaScript et pas de sélection contrairement à OpenLayers qui gère nativement la sélection d'entités : j'ai développé un système via des variables globales et événement de clic permet de sélectionner un élément et de gérer les différents comportements qui interagissent avec cette sélection (cf Figure 4.9) :

— lorsque l'on clique sur un marqueur, celui-ci est stocké dans une variable et change le marqueur pour une version jaune globale initialisée à `null` lorsqu'il n'y a aucun marqueur sélectionné,
— lorsque le panneau disparaît (soit par clic sur la carte, soit par clic sur le bouton pour fermer le panneau), la variable globale devient `null` et le style est réinitialisé,
— si le panneau est déjà ouvert (que donc un marqueur est sélectionné) et que l'utilisateur clique sur le marqueur sélectionné, le style du marqueur est réinitialisé et la variable globale devient `null`,
— si le panneau est déjà ouvert et que l'utilisateur clique sur un autre marqueur, le panneau se ferme, puis se rouvre avec les informations du nouveau marqueur sélectionné, qui est stocké dans la variable globale.

La figure 4.11 présente le code de la fonction `info` qui est appelé à chaque clic sur un marqueur de la carte. Tout d'abord si un élément est sélectionné :
— si l'utilisateur a cliqué sur le marqueur déjà sélectionné `if(selected.layer==event.layer)`, le marqueur se désélectionne, puis arrête l'exécution de la fonction.
— sinon le point se déselectionne puis continue l'exécution qui affichera les informations sur le nouveau marqueur.

La suite est une requête ajax vers la page `return_all_info_station` qui renvoit les informations sur la station avec l'identifiant passé en paramètre POST. Lorsque les informations sont récupérées :
— le panneau latéral (*sidebar*) s'affiche avec le contenu dedans (`#all_info_station`),
— l'icône du marqueur sélectionné est changée,
— et finalement on stocke l'élément sélectionné dans une variable globale.

4.2.3 Cohabitation des interactions WMS/WFS

La structure de Leaflet permet une cohabitation native des interactions avec les éléments de la carte.

4.2.4 Style

Chaque élément de la carte peut être modélisé par l'icône native de Leaflet, une icône personnalisée ou une forme (rectangle ou cercle). Les formes peuvent être modifiées à la guise de l'utilisateur en modifiant l'objet style de l'élément.

Le style est modélisé par un objet JavaScript dont les options redéfinissent le style de base de l'élément. Voici quelques exemples :
— ***stroke*** : boolean qui définit la présence d'une bordure,
— ***color*** : couleur de la bordure,
— ***fill*** : boolean qui définit si l'élément est plein,
— ***fillOpacity*** : nombre entre 0 et 1 qui représente l'opacité de l'intérieur de l'élément.

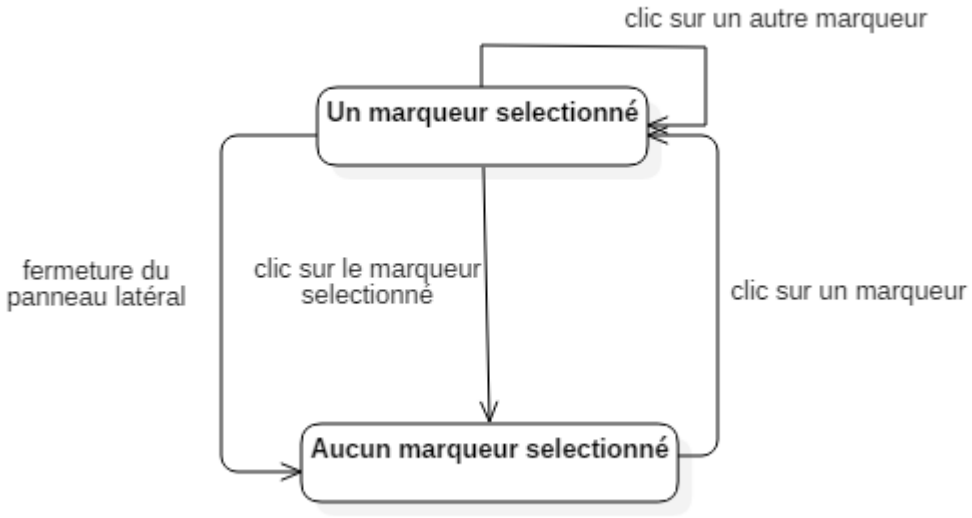

FIGURE 4.9 – Leaflet - Diagramme d'état de la sélection

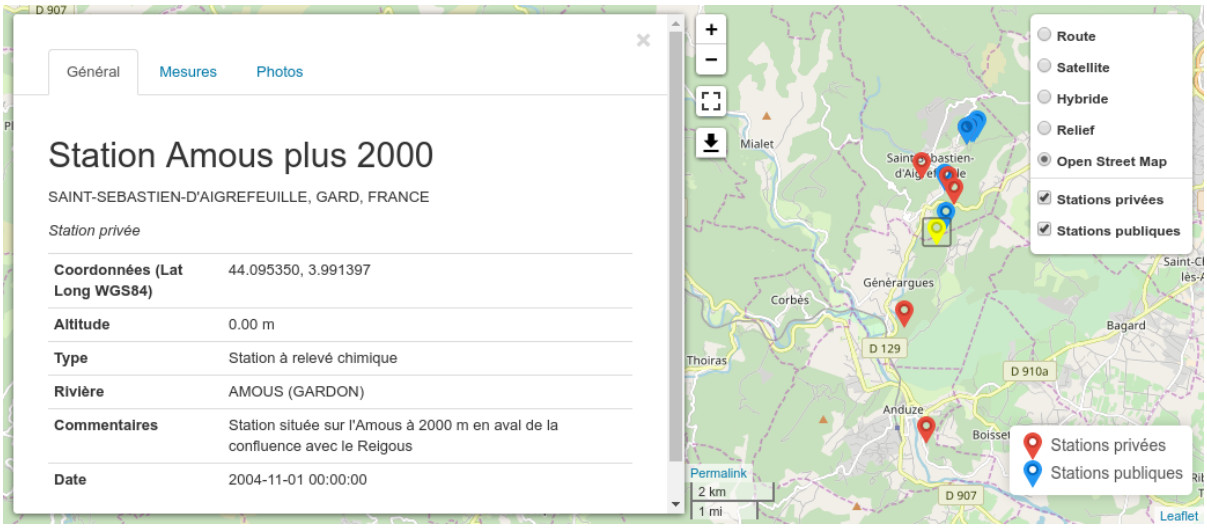

FIGURE 4.10 – Leaflet - Interactions avec la carte - WFS

```
function info(event) {
    let id = event.layer.feature.properties.st_id;
    if (selected) {
        if (selected.layer == event.layer) {
            resetStyle();
            return;
        }
        resetStyle();
    }
    $.ajax({
        type: "POST",
        url: baseurl + 'carnoules/carnoules_map/
           return_all_info_station',
        data: {station: id},
        success: function (output) {
            $('#all_info_station').html(output);
            var new_position = $('#all_info_station').offset();
            sidebar.show();
            event.layer.setIcon(selected_icon);
            selected = {};
            selected.id = id;
            selected.layer = event.layer;
        },
        dataType: 'html'
    });
}
```

FIGURE 4.11 – Leaflet - Code - Interactions avec la carte - WFS

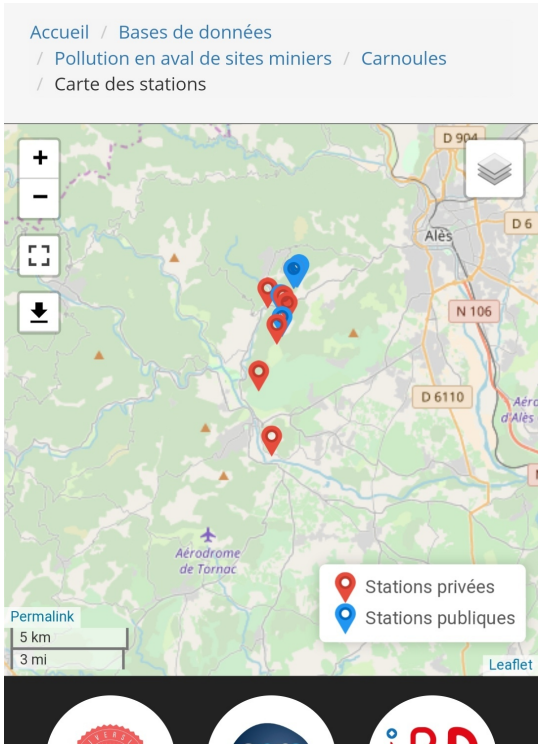

FIGURE 4.12 – Leaflet - Adaptabilité sur mobile

— ..

4.2.5 Adaptabilité

Leaflet est lui aussi *responsive*. La navigation sur la carte est très réactive et fluide. Les interactions à un ou deux doigts sont naturelles et gérées par Leaflet.

Le contrôleur *layer switcher* peut être affiché de deux manières, rétracté ou non. Sur la version web, il est affiché en entier puisque la carte est assez grande pour se permettre de le faire sans géner l'expérience utilisateur. Cependant sur un appareil plus petit, comme un téléphone, le contrôleur occupe une grande partie de l'écran, il est donc affiché de manière rétractée lorsque l'écran fait moins de 768 pixels (cf Figure 4.12).

4.2.6 Plein Écran

`https://github.com/Leaflet/Leaflet.fullscreen`

Un module qui rajoute un contrôleur sur la carte permet de passer la carte en plein écran simplement.

4.2.7 Exportation en jpg/pdf

Modules étudiés :
`https://github.com/aratcliffe/Leaflet.print`
`https://github.com/Igor-Vladyka/leaflet.browser.print`
`https://github.com/rowanwins/leaflet-easyPrint`
`https://github.com/mapbox/leaflet-image`

Leaflet propose sur son site 4 modules différents afin d'exporter une carte soit en pdf, jpg ou propose d'imprimer la carte via l'impression du navigateur. Cela marche très bien sauf quand on utilise certaines fonctionnalités comme le regroupement de points, ou des couches WMS interrogeables. A ce jour, je n'ai pas fini d'implémenter cette fonctionnalité.

4.2.8 Permalien

`https://github.com/shramov/leaflet-plugins`

Un module créé par la communauté permet de générer un permalink fonctionnel pour la carte. Ce permalien enregistre dans l'URL :

— La position de l'utilisateur sur la carte

— Le fond de carte selectionné, ainsi que les couches qui sont affichées ou non.

Cependant il ne gère pas l'enregistrement d'un élément sélectionné. Certaines recherches pourront être faites afin de savoir s'il est possible de trouver une alternative.

4.2.9 Légende

Module étudié : `https://github.com/kartoza/leaflet-wms-legend`

Un plugin, qui n'est pas sur la liste officielle du site de Leaflet, permet de rajouter une légende sur la carte. Celui ci est très proche de la solution actuelle puisqu'il créé juste un contrôleur personnalisé qui comporte une image dont l'URL est passé en paramètre à la création du plugin. Il a été fait spécifiquement pour être utilisé avec la requête `GetLegendGraphic` du protocole WMS. Or l'OREME ne veux pas utiliser cette requête qui ne satisfait pas vraiment leurs attentes à cause de son design obsolète.

4.2.10 Cartes de chaleur (Heatmap)

Module utilisé : `https://github.com/Leaflet/Leaflet.heat`
Autres modules étudiés :
`https://github.com/sunng87/heatcanvas`
`https://github.com/danielepiccone/leaflet-div-heatmap`
`https://github.com/mejackreed/leaflet-solr-heatmap`

Quatre modules ont été testés afin de sélectionner celui qui présente le meilleur équilibre entre la facilité de mise en place, et le nombre de fonctionnalités : `Leaflet.heat` est celui qui correspond le mieux aux critères.

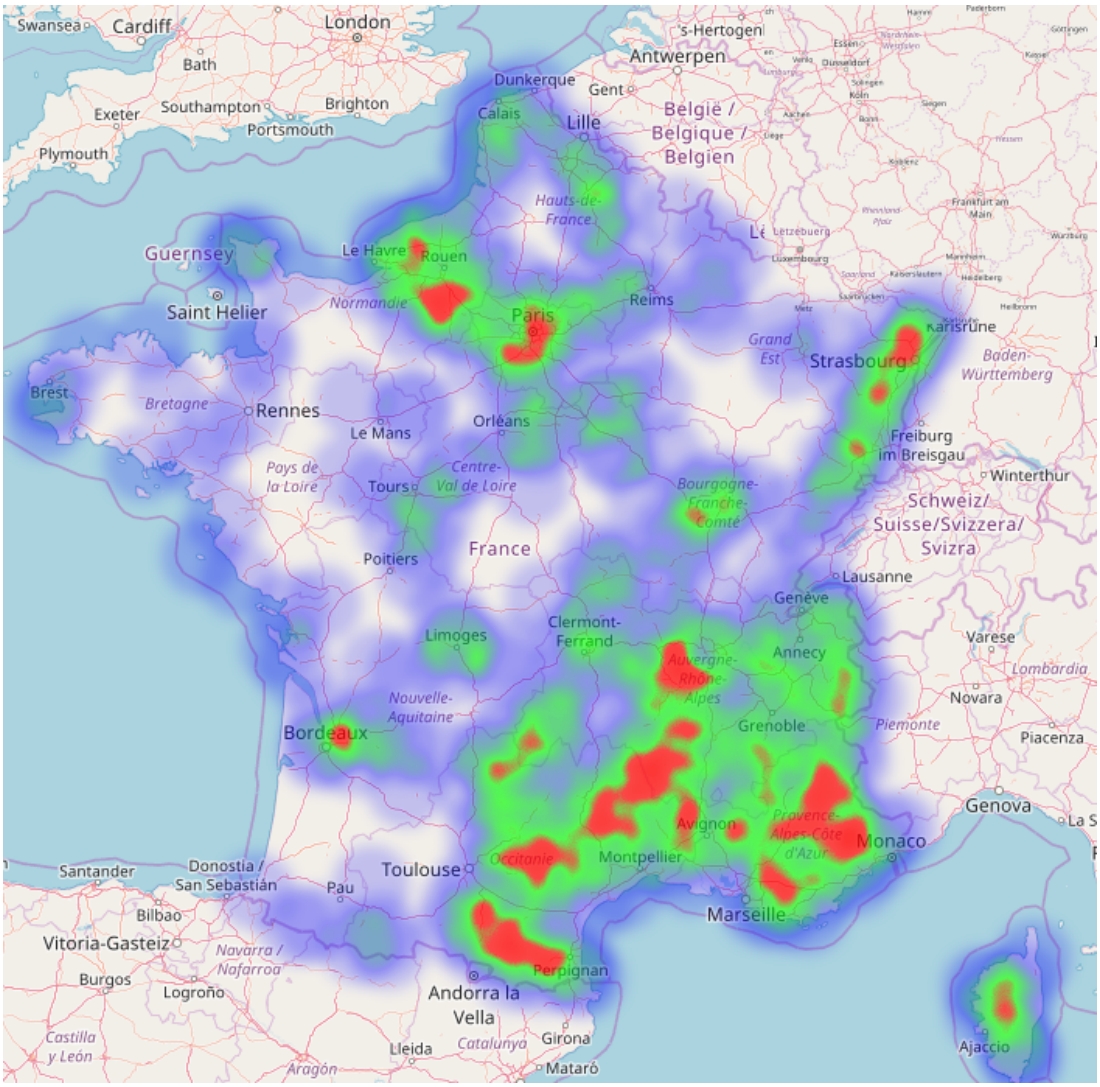

FIGURE 4.13 – Leaflet - Heatmap

Ce module utilise une autre petite bibliothèque JavaScript pour créer des heatmap dans des canvas : simpleheat. Il est possible de personnaliser le gradiant de couleur, et la chaleur d'un point (Figure 4.13).

4.2.11 Couches Temporelles

`https://github.com/socib/Leaflet.TimeDimension`

Contrairement à OpenLayers 4, Leaflet propose un module qui crée un contrôleur pour interagir avec une couche temporelle (WMS, WFS) : Figure 4.14. Ce contrôleur dispose :

— Un bouton lecture/pause,

FIGURE 4.14 – Leaflet - Contrôleur de gestion du temps

— Un bouton avance rapide et un recul rapide,
— L'affichage de la date actuellement selectionné,
— Un slider qui permet de voir l'avancée du temps dans l'amplitude temporelle proposée par la couche ainsi que d'intéragir avec elle,
— Un slider permettant de sélectionner le nombre d'images par seconde de l'animation déclenchée par le bouton play.

4.2.12 Altitude

`Leaflet` ne gère pas, tout comme OpenLayers, les couches supportant l'altitude. Cependant s'il devient indispensable de créer une carte supportant l'altitude, il est possible de créer une barre de défilement (*slider*) et de changer dans la requête l'altitude de la couche affichée par rapport à la valeur de cette barre de défilement.

4.2.13 Regroupement de points

Module utilisé : `https://github.com/Leaflet/Leaflet.markercluster`
Autre module utilisé : `https://github.com/SINTEF-9012/PruneCluster`

Un module permet de regrouper des points en un seul point s'ils sont trop proches pour s'afficher convenablement. Ce mode d'affichage aide entre autre à obtenir de meilleures performances pour le rendu d'une couche avec de nombreux points.

Les clusters sont des balises div HTML, donc stylisées avec du CSS. Ce style est moderne et le rendu général est esthétique. La couleur des markers est représenté en fonction du nombre de points que représente le *cluster*, plus il y de points, plus le point sera rouge. Contrairement à OpenLayers 4 les *cluster* sont interactifs :

— lorsque l'utilisateur passe sa souris sur le *cluster* la carte affiche un polygone qui contient tous les points du cluster afin d'avoir un idée de l'étendue du *cluster*,
— lorsque l'utilisateur clique sur le *cluster*, la carte zoome sur les tous points du cluster.
— lorsque l'utilisateur zoome ou dezoome, les *clusters* qui se regroupent en un plus gros, ou respectivement se séparent en plusieurs *clusters* sont animés ce qui rend la navigation sur la carte très fluide.

Il suffit de 2 lignes afin de créer un *layer* (`markerClusterGroup`), une pour le créer et une pour ajouter les éléments à la couche (cf Figure 4.16 et Figure 4.15).

`PruneCluster` est un autre module qui d'une manière assez proche de celle de `Leaflet.markercluster`, propose la gestion de *clusters* avec la possibilité d'intégrer des catégories dans les points. De cette manière les *clusters* affichent des informations

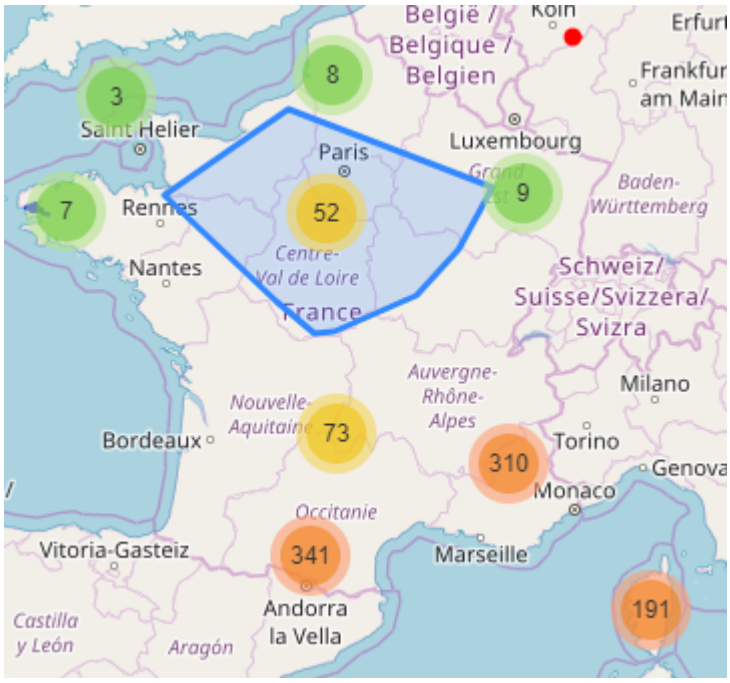

FIGURE 4.15 – Leaflet - Clusters

```
currentStations = L.markerClusterGroup().addTo(map);
currentStations.addLayer(stations);
```

FIGURE 4.16 – Leaflet - Code - Clusters

sur la proportion de points de chaque catégorie dans chaque *cluster*. Cependant je n'ai pas réussi à l'implémenter sur nos cartes.

4.2.14 Geocoding

Module utilisé : `https://github.com/perliedman/leaflet-control-geocoder`
Autres modules étudiés :
`https://github.com/smeijer/L.GeoSearch`
`https://github.com/k4r573n/leaflet-control-osm-geocoder`
`https://github.com/Esri/esri-leaflet-geocoder`
`https://github.com/OpenCageData/leaflet-opencage-search`

Un grand nombre de modules sur Leaflet permettent de faire du geocoding. Après en avoir testé plusieurs, celui qui est utilisé fonctionne en rajoutant un contrôleur sur la carte. Il présente trois avantages par rapport aux autres modules :

— il est très simple de l'instancier et de l'ajouter à la carte,
— les textes (messages d'erreurs, placeholder) sont configurables afin de les traduire,
— le zoom lors d'un recherche d'une zone (pays, ville, région, ..) englobe la zone et ne

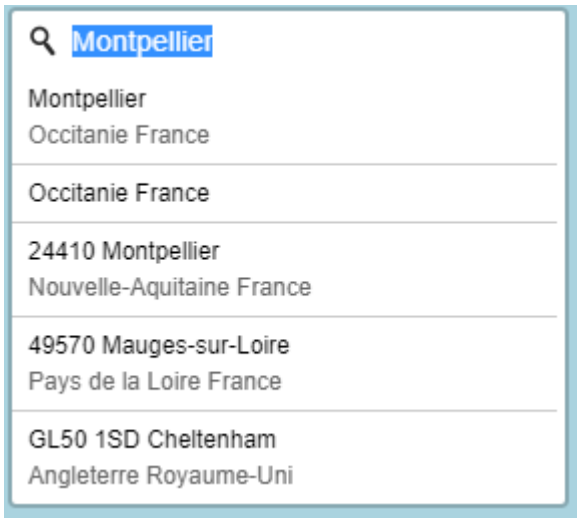

FIGURE 4.17 – Leaflet - Contrôleur Geocoder

```
1 L.Control.geocoder({
2 placeholder: "Recherche..",
3 errorMessage: "Aucun resultat.",
4 expand: "click",
5 collapsed: "true",
6 position: "topright"
7 }).addTo(map);
```

FIGURE 4.18 – Leaflet - Code - Géocodage*

zoome pas sur le centre de la zone recherchée contrairement à celui d'OpenLayers 4 (Figure 4.17).

Dans la figure 4.18, un nouveau contrôleur de Géocodage* est instancié et ajouté à la carte. Son seul paramètre est un objet avec les différentes options pour le configurer : quelques traductions, l'affichage du contrôleur via un bouton qui s'étend vers un champ de saisie lors du clic (options `expand`et `collapsed`), et on positionne le bouton en haut à droite de la carte.

4.2.15 Recherche par propriété

`https://github.com/stefanocudini/leaflet-search/`

Un module permet de faire une recherche selon les attributs d'un élément (feature) GeoJson*. Par exemple, cela donne la possibilité à l'utilisateur de chercher une station de mesure par son nom et de zoomer vers sa position sur la carte (cf Figure 4.20).

Ce module de recherche est très complet, il propose beaucoup d'options diverses afin de personnaliser le comportement de la recherche (traduction, icône à afficher, nombre

```
searchControl = new L.Control.Search({
layer: currentStations,
propertyName: "station",
marker: false,
moveToLocation: function(latlng, title, map) {
    let zoom = 17;
    map.flyTo(latlng, zoom);
}
}).addTo(map);
```

FIGURE 4.19 – Leaflet - Code - Recherche par propriété

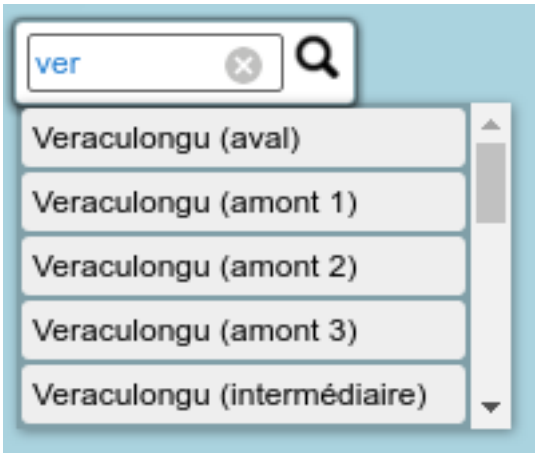

FIGURE 4.20 – Leaflet - Recherche par propriété

de caractère minimum pour activer l'autocomplétion et plus encore).

Cependant, deux fonctionnalités manquent à ce module :

— la possibilité de combiner la recherche de Géocodage* et celle ci dans un seul contrôleur, puisque si la carte doit supporter le Géocodage* et la recherche par propriété, deux contrôleurs de recherche seront donc sur la carte. Cela peut embrouiller un utilisateur non averti.
— le module propose uniquement de rechercher des éléments en recherchant via une seule propriété d'une seule couche de la carte.

Figure 4.19 : un nouveau contrôleur de recherche est instancié. La recherche s'effectue sur le layer `currentStations` (stations affichés actuellement sur la carte), la propriété recherché est `station` (le nom de la station). La fonction `moveToLocation` gère le zoom vers l'élément séléctionné, ici j'utilise la méthode `flyTo` de l'objet `L.map` qui fera une animation de zoom vers les coordonnées et le niveau de zoom fournit en paramètre.

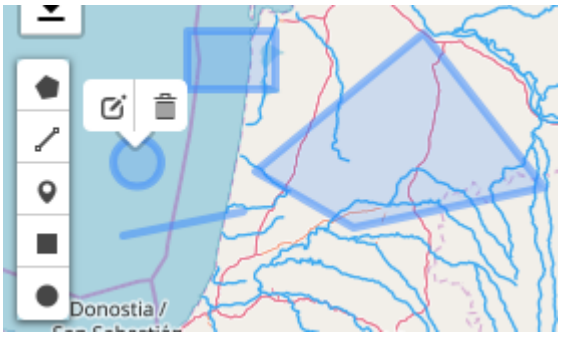

FIGURE 4.21 – Leaflet - Annotation graphique

```
let drawnItems = new L.FeatureGroup().addTo(map);
new LeafletToolbar.DrawToolbar({
position: 'topleft',
}).addTo(map);
map.on('draw:created', function(evt) {
let     type = evt.layerType,
    layer = evt.layer;
drawnItems.addLayer(layer);
layer.on('click', function(event) {
    new LeafletToolbar.EditToolbar.Popup(event.latlng, {
        actions: editActions
    }).addTo(map, layer);
});
});
```

FIGURE 4.22 – Leaflet - Code - Annotation graphique

4.2.16 Annotation graphique

Module utilisé : `https://github.com/justinmanley/leaflet-draw-toolbar`
Dépendance : `https://github.com/Leaflet/Leaflet.toolbar`
Autres modules étudiés :
`https://github.com/justinmanley/Leaflet.Illustrate`
`https://github.com/tcoupin/leaflet-paintpolygon`
..

Parmi les modules de dessin pour Leaflet, un semble être plus évolué : Il propose la modification et la suppression des éléments dessinés via un pop-up qui s'affiche lorsque l'on clique sur l'élément que l'on veut modifier (Figure 4.21). De plus, un large choix d'éléments à dessiner est proposé : polygones, rectangles, lignes, points, cercles.

Cependant, il est impossible de modifier les textes de description des outils, il faudra donc modifier le code source du module soi-même. Ces modifications seront à renouveler à chaque mise à jour du module.

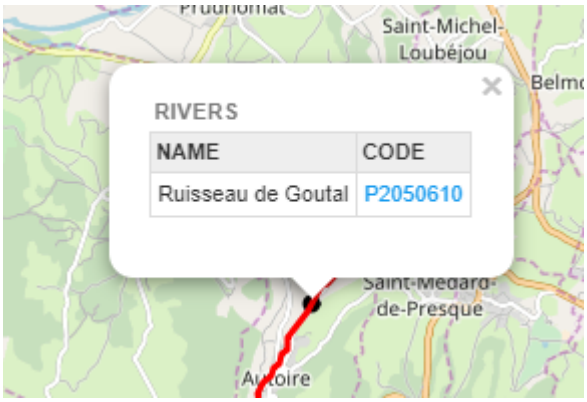

FIGURE 4.23 – Leaflet - Pop-Up

```
L.WMS.Source = L.WMS.Source.extend({
...
'showFeatureInfo': function (latlng, info) {
    if (info.includes('</table>')) {
        a = L.popup();
        a.setLatLng(latlng);
        a.setContent(info);
        a.openOn(map);
        a.on("remove", resetStyleFleuve);
    }
},
...
});
```

FIGURE 4.24 – Leaflet - Code - Popup

Figure 4.22 : `LeafletToolbar` nécessite des efforts d'implémentation afin de fonctionner :

- instancer une couche (*layer*) et l'ajouter à la carte. Ce *layer* sera utilisé pour ajouter chaque annotation graphique,
- ajouter le contrôleur qui comporte les boutons avec tous les éléments à créer sur la carte,
- configurer l'affichage d'un pop-up avec les deux actions de modifier ou de supprimer d'une annotation graphique.

4.2.17 Pop-Up

Leaflet dispose d'une classe qui permet de créer un pop-up très facilement sur la carte en lui fournissant un contenu (DOM ou texte/html) et une position (lat/lng). La méthode `open` permet d'afficher le pop-up sur la carte : Figure 4.23 Ce morceau de

FIGURE 4.25 – Leaflet - Labellisation des points

```
function affTooltip() {
currentStations.eachLayer(function (e) {
    e.bindTooltip(e.feature.properties.station, {permanent: true
       });
})
}
function enleverTooltip() {
currentStations.eachLayer(function (e) {
    e.unbindTooltip();
})
}
```

FIGURE 4.26 – Leaflet - Code - Fonctions d'affichage des labels

code (Figure 4.24) provient de la fonction qui gère l'affichage des données récupérées lors d'un clic sur une entité hydrographique. Si la réponse (texte) est composée d'un tableau, un popup est créé `L.popup()`, on lui assigne une coordonnée, le texte, puis on l'affiche sur la carte.

4.2.18 Labellisation des points

Les info-bulles (*tooltip*) sont des objets que l'on peut attribuer à n'importe quel élément de la carte, dont les stations. Ils permettent d'afficher du contenu textuel, soit lorsque la souris se trouve sur le point en question, soit de l'afficher de manière permanente (Figure 4.25).

Figure 4.26 : Afin de labelliser les points qui décrivent chaque station (currentStations), j'ai créé 2 fonctions, une qui attache l'infobulle (bindTooltip) avec le nom de la station (e.feature.properties.station) et une autre qui enlève ce tooltip (unbindTooltip).

4.3 Synthèse

Importance	Fonctionnalité	OpenLayers	Leaflet
Indispensable	WMS	Natif	Module
	Interactions WMS	A nécessité un développement	Module, amélioré pour le proxy
	WFS	Natif	Natif
	Interactions WFS	Natif	A nécessité un développement
	Cohabitation des interactions WMS/WFS	Natif	Natif
	Exportation de la carte	Natif	A finir
	Permalien	A nécessité un développement, gère aussi la sélection d'entité	Module simple, mais ne gère pas la sélection d'entité
	Filtre	Natif	Natif
	Pop-Up	A nécessité un développement	Natif
	Labellisation des points	Natif	Natif
Additionnel	Heatmap	Natif	Module
	Légende	Non	Non
	Temporelle	A nécessité un développement, fonctionne de manière basique	Module très complet
	Altitude	Non	Non
	Géocodage*	Module	Module complet
	Recherche par propriété	Non	Module
	Regroupement de points	Natif pas complet	Module complet + possibilité de catégoriser les clusters
	Annotation graphique	Natif pas complet	Module complet (modification, suppresion)

Leaflet propose les mêmes fonctionnalités qu'OpenLayers, certaines sont mêmes mieux gérées avec Leaflet, d'autres non. OpenLayers propose par exemple un support natif de la séléction d'entités contrairement à Leaflet pour lequel j'ai du développer

une solution, les permaliens permettent de partager aussi l'élément actuellement sélectionné contrairement à Leaflet. L'exportation ne pose quant à elle aucun problème à OpenLayers grâce à l'unique utilisation de canvas.

Cependant, Leaflet propose des équivalents, ou que j'ai pu compléter pour atteindre le même niveau d'exigence que pour OpenLayers. Et il gère d'autres fonctionnalités, comme l'interaction avec une couche WMS simplement, un support complet des *clusters*, des couches temporelles, du géocodage et de la recherche par une propriété, et aussi les annotation graphiques.

Compte tenu des résultats de l'analyse et de cette comparaison, Leaflet semble être plus adapté aux besoins de l'OREME. De plus la bibliothèque est plus légère et simple à utiliser qu'OpenLayers : en comparant les cartes faites avec OpenLayers 4 et Leaflet, OpenLayers 4 prend 2,5 fois plus de lignes pour implémenter autant de fonctionnalités.

D'autre part, la communauté de Leaflet est maintenant bien plus importante et active que celle d'OpenLayers, elle créé et tient à jour de nombreux module pour Leaflet. Le domaine de la cartographie est encore en pleine évolution, cet aspect modulaire de Leaflet est donc intéressant grâce à la participation active de la communauté, ce qui permettra d'intégrer de nouvelles fonctionnalités bien plus rapidement que sur une bibliothèque comme OpenLayers.

4.4 Perspectives

Compte tenu de mon avancement, certaines tâches de ma mission ne sont pas encore terminées, avant la fin du stage, j'aimerais finir d'implémenter :

- finaliser la migration des cartes des deux cartes (exportation, détails de style..),
- conception d'un utilitaire pour simplifier la création des cartes et évider les redondances dans le code,
- tester l'utilitaire et l'intégration des modules entre eux,
- créer un module de génération automatique de légende.

Chapitre 5

Rapport d'activité

5.1 Organisation

Au début de mon stage, n'ayant jamais pratiqué la cartographie Web et les standards du domaine, je n'avais pas établi de plan de travail précis ; mais seul sur ce projet, j'ai eu la liberté de m'organiser au fur et à mesure de l'avancement du travail.

En plus, avec les fonctionnalités qui ont été rajoutées au cahier des charges, il m'était difficile de concevoir une organisation prévisionnelle. Je ne pouvais donc pas savoir le temps qu'allait me prendre la migration des deux cartes. J'ai donc choisi une planification journalière.

Chaque fin de journée, je me préparais une *todo list* récapitulant les fonctionnalités à implémenter, objectifs à réaliser et problèmes à résoudre pour les jours suivants.

Tout au long du projet, j'ai pris des notes sur mes avancées et toutes les informations que j'ai jugées importantes.

5.2 Réunions

De manière assez régulière, nous faisions des comptes rendu de mon avancée assez souvent, soit lorsque j'avais beaucoup de contenu à présenter, soit lorsque j'avais des questions et précisions sur le cahier des charges, soit pour proposer de nouvelles fonctionnalités que je trouvais durant mes recherches.

5.3 Outils

Afin de réaliser la mission, un certain nombre d'outils ont été utilisés.

5.3.1 Subversion

Le site web de l'Oreme est versionné à l'aide de SVN. Une branche de développement a été créé spécialement pour la migration ainsi qu'un sous domaine du site qui me permettait de tester les cartes et les nouvelles fonctionnalités sur d'autres appareils (smartphone, ordinateur peu puissant ou faible connexion).

5.3.2 PG Admin

PG Admin est un logiciel d'administration de base de données PostgreSQL qui a été utilisé pour consulter le contenu de la base de données.

5.3.3 PhpStorm

PhpStorm est un IDE de l'entreprise Jetbrains très complet qui gère le PHP, HTML, CSS et JavaScript. C'est l'IDE qui a été utilisé pour le développement des cartes.

5.4 Problèmes rencontrés

Durant le projet, je me suis heurté à quelques problèmes de programmation :

— Il m'arrivait de confondre certaines spécifications ou méthodes d'une bibliothèque avec l'autre puisque je travaillais sur les deux en même temps,

— j'ai eu tendance à m'égarer trop longtemps sur des erreurs que je n'arrivais pas à résoudre. Il m'arrivait quelques fois de ne pas réussir à prendre assez de recul pour changer de tâche et y revenir plus tard

Dans le futur, j'aimerais ne plus bloquer longtemps sur un problème pour faire une autre tâche ce qui me permettrait de perdre moins de temps.

Chapitre 6

Conclusion

Ce projet avait pour but de remplacer, dans le portail de données de l'OSU OREME, une ancienne version de bibliothèque de cartographie web par une nouvelle plus dans l'air du temps, plus facile à maintenir, qui propose de nouvelles fonctionnalités sans perdre les fonctionnalités actuelles. Une autre contrainte était l'ergonomie et la responsivité du site qui était à améliorer et rendre les cartes agréables et fluides à utiliser sur téléphone. Pour cela j'ai migré les deux cartes avec Leaflet et OpenLayers avant de comparer et analyser chaque fonctionnalité, le comportement de ces bibliothèques, les points forts et points faibles, et la simplicité pour migrer vers cette nouvelle librairie.

Au final, la bibliothèque Leaflet a été retenue car c'est celle qui correspond le mieux aux besoins actuels et futurs de l'OREME, le fonctionnement de chaque fonctionnalité est expliqué dans ce rapport et une carte présentant chacune des fonctionnalités ont été implémentées, commentées et documentées. En termes de perspective, la conception d'un utilitaire pour simplifier la mise en place de nouvelles cartes est encore à faire, et est nécessaire pour finir la migration.

Ce projet fut une expérience complète dans le domaine de la programmation Web côté client, j'ai utilisé des librairies d'un domaine qui m'était jusque là encore inconnu : la cartographie. Durant ce stage, j'ai beaucoup appris, tout d'abord sur le développement en JavaScript, mais aussi sur le travail en autonomie et surtout sur la veille informatique, que j'ai beaucoup pratiqué pendant ce projet et j'ai l'intention de continuer de me livrer à cette exercice. J'ai aussi beaucoup apprécié le fait de travailler sur un projet 'réel', un projet qui va vraiment être utilisé par des chercheurs dans un but précis. Cela m'a beaucoup motivé à produire le meilleur de moi-même.

Bibliographie

[1] (). Présentation — indigeo - INfrastructure de Données et d'Informations GEOréférencées sur l'environnement, adresse : http://indigeo.fr/ (visité le 10/06/2018).

[2] (). Leaflet — an open-source JavaScript library for interactive maps, adresse : https://leafletjs.com/ (visité le 10/06/2018).

[3] (). OpenLayers - Welcome, adresse : http://openlayers.org/ (visité le 10/06/2018).

[4] (). Components · Bootstrap, adresse : https://getbootstrap.com/docs/3.3/components/ (visité le 10/06/2018).

[5] (). Welcome to CodeIgniter : CodeIgniter User Guide, adresse : https://codeigniter.com/userguide2/ (visité le 10/06/2018).

[6] Iconfinder. (). Map icons - download 30230 free & premium icons, Iconfinder, adresse : https://www.iconfinder.com/search/ (visité le 10/06/2018).

[7] (). SlideMenu Demo, adresse : http://unbam.github.io/Leaflet.SlideMenu/ (visité le 10/06/2018).

[8] (). sidebar-v2 example, adresse : https://turbo87.github.io/sidebar-v2/examples/ol3.html#home (visité le 10/06/2018).

[9] (). jonataswalker/ol-geocoder : Geocoder Nominatim for OpenLayers, adresse : https://github.com/jonataswalker/ol-geocoder (visité le 10/06/2018).

[10] S. Cudini, *leaflet-search : Search stuff in a Leaflet map*, original-date : 2012-08-06T21 :15 :55Z, 6 juin 2018. adresse : https://github.com/stefanocudini/leaflet-search (visité le 10/06/2018).

[11] *PruneCluster : Fast and realtime marker clustering for Leaflet*, original-date : 2014-05-06T13 :02 :08Z, 7 juin 2018. adresse : https://github.com/SINTEF-9012/PruneCluster (visité le 10/06/2018).

[12] *leaflet-wms-legend : A simple leaflet WMS legend widget*, original-date : 2014-08-24T13 :27 :33Z, 25 mai 2018. adresse : https://github.com/kartoza/leaflet-wms-legend (visité le 10/06/2018).

[13] *leaflet.wms : A Leaflet plugin for working with Web Map services, providing : single-tile/untiled/nontiled layers, shared WMS sources, and GetFeatureInfo-powered identify*, original-date : 2014-10-03T21 :44 :24Z, 27 mai 2018. adresse : https://github.com/heigeo/leaflet.wms (visité le 10/06/2018).

[14] (). WMS output formats — GeoServer 2.13.x User Manual, adresse : http://docs.geoserver.org/stable/en/user/services/wms/outputformats.html#wms-output-formats (visité le 10/06/2018).

[15] (). WMS reference — GeoServer 2.13.x User Manual, adresse : http://docs.geoserver.org/stable/en/user/services/wms/reference.html#wms-getcap (visité le 10/06/2018).

[16] (). GeoServer User Manual — GeoServer 2.13.x User Manual, adresse : http://docs.geoserver.org/stable/en/user/index.html (visité le 10/06/2018).

[17] *Leaflet.Sleep : Prevent unwanted scroll capturing ; let you map sleep*, original-date : 2014-09-22T05 :14 :58Z, 8 juin 2018. adresse : https://github.com/CliffCloud/Leaflet.Sleep (visité le 10/06/2018).

[18] P. Liedman, *leaflet-control-geocoder : A simple geocoder form to locate places. Easily extended to multiple data providers*, original-date : 2013-09-27T10 :44 :00Z, 7 juin 2018. adresse : https://github.com/perliedman/leaflet-control-geocoder (visité le 10/06/2018).

[19] D. Piccone, *leaflet-div-heatmap : Heatmap layer for leaflet using CSS radial gradients and divIcon class*, original-date : 2013-10-07T17 :06 :26Z, 6 juin 2018. adresse : https://github.com/danielepiccone/leaflet-div-heatmap (visité le 10/06/2018).

[20] I. Vladyka, *leaflet.browser.print : A leaflet plugin which allows users to print the map directly from the browser*, original-date : 2017-05-17T10 :47 :08Z, 23 mai 2018. adresse : https://github.com/Igor-Vladyka/leaflet.browser.print (visité le 10/06/2018).

[21] (). How to Write a Thesis in LaTeX pt 1 - Basic Structure, adresse : https://fr.sharelatex.com/blog/2013/08/02/thesis-series-pt1.html (visité le 10/06/2018).

[22] (). Mémento, OpenClassrooms, adresse : https://openclassrooms.com/courses/redigez-des-documents-de-qualite-avec-latex/memento-2 (visité le 10/06/2018).

[23] (). Leaflet vs Mapbox vs OpenLayers 2018 Comparison, StackShare, adresse : https://stackshare.io/stackups/leaflet-vs-mapbox-vs-openlayers (visité le 10/06/2018).

[24] P. Shramov, *leaflet-plugins : Plugins for Leaflet library*, original-date : 2012-04-18T17 :19 :20Z, 1er juin 2018. adresse : https://github.com/shramov/leaflet-plugins (visité le 10/06/2018).

[25] T. Bieniek, *leaflet-sidebar : A responsive sidebar for Leaflet maps*, original-date : 2013-10-29T19 :29 :09Z, 9 juin 2018. adresse : https://github.com/Turbo87/leaflet-sidebar (visité le 10/06/2018).

[26] *Leaflet.TimeDimension : Add time dimension capabilities on a Leaflet map.* original-date : 2014-12-19T11 :59 :28Z, 7 juin 2018. adresse : `https://github.com/socib/Leaflet.TimeDimension` (visité le 10/06/2018).

[27] *Leaflet.heat : A tiny, simple and fast heatmap plugin for Leaflet*, original-date : 2014-01-31T13 :34 :50Z, 10 juin 2018. adresse : `https://github.com/Leaflet/Leaflet.heat` (visité le 10/06/2018).

[28] N. Sun, *heatcanvas : Pixel based heatmap with html5 canvas*, original-date : 2011-05-26T02 :03 :36Z, 7 juin 2018. adresse : `https://github.com/sunng87/heatcanvas` (visité le 10/06/2018).

[29] J. Reed, *leaflet-solr-heatmap : A Leaflet plugin that visualizes heatmap facets from Solr*, original-date : 2015-06-29T14 :26 :40Z, 18 avr. 2018. adresse : `https://github.com/mejackreed/leaflet-solr-heatmap` (visité le 10/06/2018).

[30] (). Welcome to the OGC | OGC, adresse : `http://www.opengeospatial.org/` (visité le 10/06/2018).

[31] *Leaflet.markercluster : Marker Clustering plugin for Leaflet*, original-date : 2012-07-11T01 :54 :39Z, 7 juin 2018. adresse : `https://github.com/Leaflet/Leaflet.markercluster` (visité le 10/06/2018).

[32] S. Meijer, *leaflet-geosearch : Leaflet geosearching/geocoding control*, original-date : 2012-12-24T01 :18 :58Z, 6 juin 2018. adresse : `https://github.com/smeijer/leaflet-geosearch` (visité le 10/06/2018).

[33] Karsten, *Contribute to leaflet-control-osm-geocoder development by creating an account on GitHub*, original-date : 2013-03-15T15 :47 :33Z, 16 mai 2018. adresse : `https://github.com/k4r573n/leaflet-control-osm-geocoder` (visité le 10/06/2018).

[34] *esri-leaflet-geocoder : API helpers and UI controls for geocoding with the ArcGIS Geocoding Service with Leaflet*, original-date : 2013-11-04T18 :24 :46Z, 6 juin 2018. adresse : `https://github.com/Esri/esri-leaflet-geocoder` (visité le 10/06/2018).

[35] *leaflet-opencage-search : A Leaflet search control that uses OpenCage Data's geocoder*, original-date : 2014-04-17T16 :03 :40Z, 18 mai 2018. adresse : `https://github.com/OpenCageData/leaflet-opencage-search` (visité le 10/06/2018).

[36] J. Manley, *Leaflet.toolbar for Leaflet.draw. Example : https ://justinmanley.github.io/leaflet-draw-toolbar/examples/popup.html*, original-date : 2017-09-20T03 :18 :21Z, 10 juin 2018. adresse : `https://github.com/justinmanley/leaflet-draw-toolbar` (visité le 10/06/2018).

[37] *Leaflet.toolbar : Flexible, extensible toolbars for Leaflet maps*, original-date : 2014-09-22T22 :36 :34Z, 10 juin 2018. adresse : `https://github.com/Leaflet/Leaflet.toolbar` (visité le 10/06/2018).

[38] ——, *Leaflet.Illustrate : Rich annotation plugin (drawing, text, and more) for Leaflet. Designed to help people tell the story behind their maps*, original-date : 2014-06-23T17 :27 :36Z, 4 juin 2018. adresse : `https://github.com/justinmanley/Leaflet.Illustrate` (visité le 10/06/2018).

[39] *leaflet-paintpolygon : Leaflet plugin to create polygon with circle as paint*. adresse : `https://github.com/tcoupin/leaflet-paintpolygon` (visité le 10/06/2018).

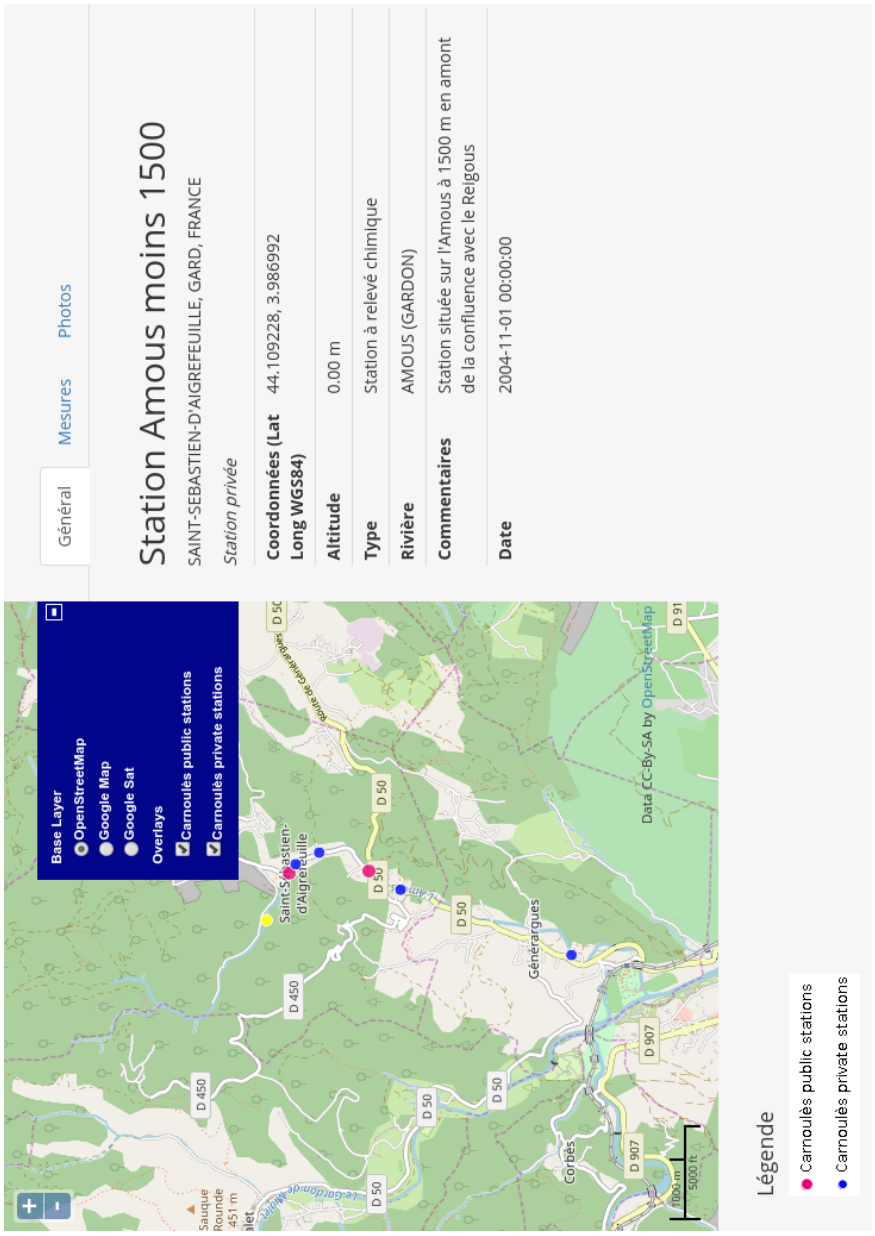

FIGURE 1 – Carte de Carnoulès - OpenLayers 2

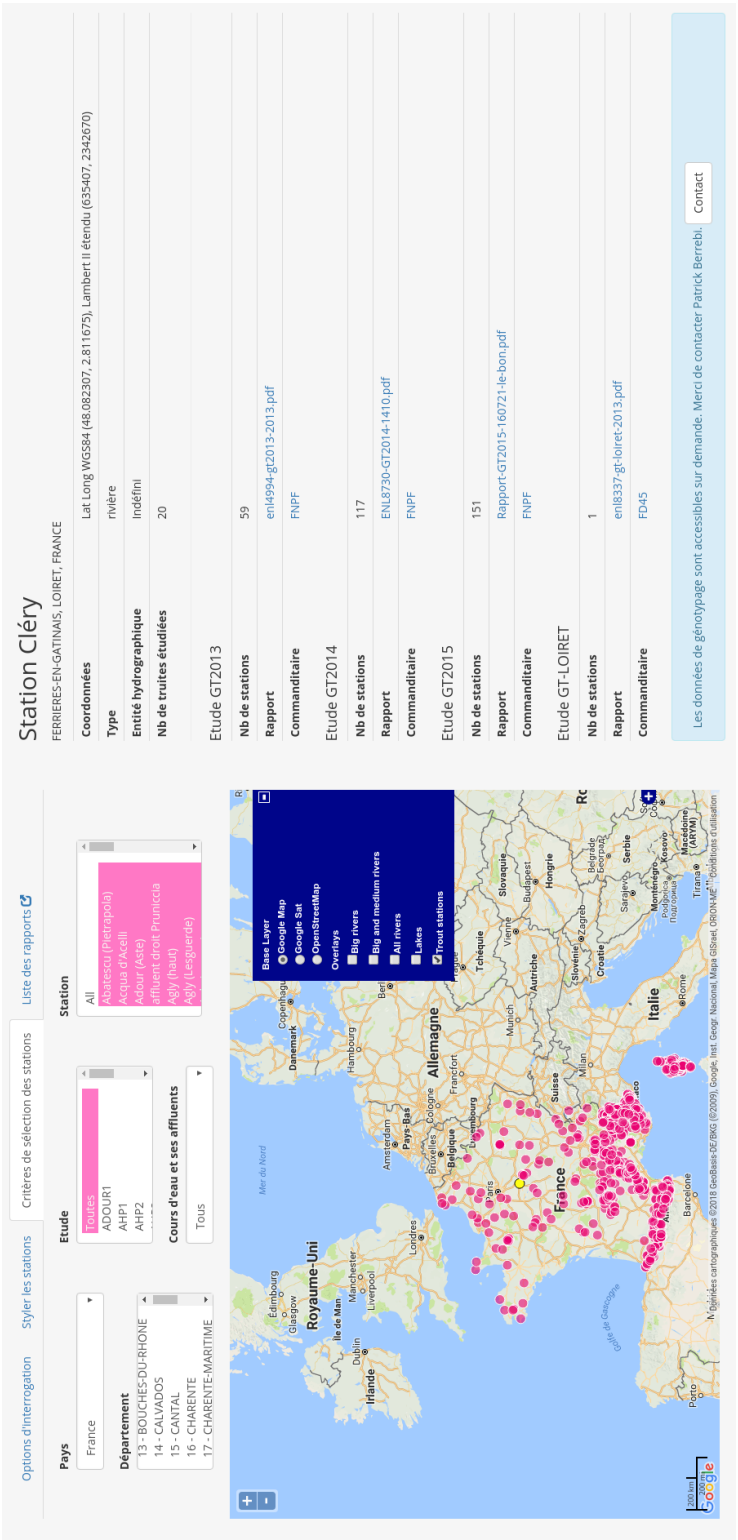

FIGURE 2 – Carte des truites communes - OpenLayers 2

```
if (window.location.hash !== '') {
    var hash = window.location.hash.replace('#map=', '');
    var parts = hash.split('/');
    if (parts.length === 6) {
        // Zoom de la carte
        zoom = parseInt(parts[0], 10);
        // Centre de la carte
        center = [
            parseFloat(parts[1]),
            parseFloat(parts[2])
        ];
        // Rotation de la carte
        rotation = parseFloat(parts[3]);
        // Si un point est selectionne
        if (parts[4] != 'null') {
            // cf Figure suivante
        } else {
            // S'il n'y a pas de stations selectionne
            selected = null;
            $('#affPanel').attr("disabled", true);
        }
        // Fond de carte
        idlayerBase = parts[5];
    }
}
```

FIGURE 3 – OpenLayers 4 - Code - Permalink - Récupération des données (1)

```
1  // Identifiant de la station selectionnee
2  selected = parts[4];
3  // Recuperation des informations sur la stations
4  $.ajax({
5      type: "POST",
6      url: baseurl + 'carnoules/carnoules_map/
          return_all_info_station',
7      data: {station: parts[4]},
8      success: function (output) {
9          $('#all_info_station').html(output);
10         var new_position = $('#all_info_station').offset();
11         window.scrollTo(new_position.left, new_position.top);
12     },
13     dataType: 'html'
14 });
15 $('#affPanel').click(sortirPanel);
16 // Lorsque les stations sont recuperees :
17 Promise.all([privateFetch, publicFetch]).then(function () {
18     let features = privateSource.getFeatures().concat(
          publicSource.getFeatures());
19     selectedF = features.find(function (f) {
20         if (f.getProperties().st_id.includes(selected)) return f;
21     });
22     if (selectedF) {
23         // Donne le style de selection a la station.
24         selectedF.setStyle(selectStyle);
25         // EventListener pour deselection la station au clic sur
              la carte
26         $("#map").one('click', deSelectFeature)
27     }
28 });
```

FIGURE 4 – OpenLayers 4 - Code - Permalink - Récupération des données (2)

```
window.addEventListener('popstate', function (event) {
    if (event.state === null) {
        return;
    }
    map.getView().setCenter(event.state.center);
    map.getView().setZoom(event.state.zoom);
    map.getView().setRotation(event.state.rotation);
    shouldUpdate = false;
});
```

FIGURE 5 – OpenLayers 4 - Code - Permalink - Retour en arrière